- If underlying process model is lucent & event log is translucent: we can discover the exact process model using the log quite easily!
- Are lucent process models even useful or prevalent? Yes, short-circuited sound workflow-nets are lucent, which means we could apply our assumption of lucent process models for the majority of business process models. This is why the access to a translucent event log is crucial for simple & accurate process modelling.
- For testing: 1. Randomly generate a sound workflow net (with extra conditions, so this would probably be something like a data petri net or even a stochastic data petri net)or take a couple from preexisting examples (maybe real-life examples?). 2. Generate translucent event logs and normal event logs by playing out the model. 3. Use the developed method to generate translucent event log from the normal event logs. 4. Compare each of the logs!
- Definition of lucency ($en(AM, s_0) = en(AM, s_1) \Rightarrow s_0 = s_1$): looks similar to the definition of injective functions. Would it also make sense to define a class of automata that fulfills the definition of surjection & bijection? Surjection would be: $\forall M \in$ Set of activities$\exists s : en(AM, s) = M$, a bijection would mean that for every state, there is a one-on-one mapping to the power set of all activities. That wouldn't bring us too far.
- Does using neural networks (e.g. transformers) to predict activated events even make sense? We're doing all this to create a good, representative process model, why not just let the transformer predict the model directly? (You know this sounds pretty dumb right) No, what I'm arguing is that "generating" the set of activated activities using a neural network goes not guarantee any correctness of it (you might find an activity in the generated translucent event log which is not backed up by the event log progression). Using classical, restrictive machine learning techniques might be work better? I'm not sure though.
- Would it make sense to restrict our study to sound workflow nets? I mean, most practical usages will be used for these process models. If we assume that generating translucent logs are primarily used for process discovery of lucent process models, generalizing our method to logs of potential non-lucent process models wouldn't be that useful at all. We might discover some other classes of process models where this might be useful, but mal gucken.
- I really want to push the notion of transparent - translucent - (opaque) event logs if translucent logs are already a thing. Transparent: We can directly generate a correct process model out of the log (state & enabled activities are in there, in another paper it was also called super-translucency or something), and opaque would be just normal logs.
- In this paper, we observe two cases: 1. We only have the event log available and 2. We have an event log and the corresponding (underfitting, of course) process model. Is the process model discovered or taken from the business model? (Can we assume that the model is mostly correct?) That's trivial, since we can always use our current process discovery algorithm to generate a semi-correct model. We can safely assume that the model is einigermaßen correct. We can also add a catchy name to this part, something like top-down vs. bottom-up or build vs. trim approaches. The

inductive miner already produces a sound workflow net, i.e. a lucent model. So... if we already are assuming that the underlying process model is lucent... Why do we need the translucent logs anyway? Well, the limitations of the inductive miner are that it tends to return a rather underfitting model, so generating (highly correct) translucent logs will probably help discover a more fitting model.

- Another difficulty in using neural networks to generate translucent event logs is that neural networks necessitate label data. This is a chicken or the egg dilemma. The paper "Creating Translucent Event Logs to Improve Process Discovery" tries to solve this problem by providing labels to patterns which are in a snapshot which relates to a certain state, or just looking at the given process model and adding every enabled activities.

- The first evaluation process works like this: Given a normal process log, we use state-of-the-art process discovery algorithms (Split Miner, Inductive Miner) to generate process models. We then use the log as input to our program and generate a translucent event log. We then generate a process model based on translucent-log based process discovery algorithms. I don't really know if there are any or what they're called. Ask Harry for pointers. Answer: Not published yet. Might have to wait for publication. We then compare the models based on their performance measures. A similar method would be the case for the top-down method, where we get a discovered model as input: We would then just skip the process discovery step and the rest would be analogous.

- The second evaluation process works like this: With the PLG library, we generate random process models, e.g. data Petri nets (Not sure if that is possible using pure PLG, some modifications might be needed). We then play-out the model either using PLG or our own program randomly and extract 1. a normal log and 2. a translucent event log. We then use the normal log as input to our Translucent-Log-Generation-Program and compare the result with the original translucent log.

- Limitations: Using real-life event logs is often not plausible for our scenario due to the fact that most real-life logs do not contain the set of enabled activities and are therefore not translucent. I couldn't find a way yet to integrate real-life event logs for this work, and we will most likely use synthetically generated event logs for our thesis.

- Another good evaluation method would be

- Precision by discovering long term dependencies.

- Next steps: Develop a library to play-out translucent logs from a process model (petri net).

- use pm4py instead of prom for webapp development

- In a process, there is stateful data involved. The data might fluctuate independent of occurred events, but we limit the data states only to a discrete time interval: Before, between and after two events. It is logical to observe these kinds of data only due to the intrinsic nature of event logs: they are produced right after some event has been executed by e.g. an IoT device. There are several ways to denote this flow of data state in combination with the Petri Net setting. The data petri net does not

necessarily denote how data changes througout the process flow, but only dictates the probability distribution of individual transitions for each token with data event. Colored Petri Nets, however, impertively determines the data flow by using guards, arc functions and colors.

- Note that translucent logs is mainly useful for that case of a lucent process model. We can also create a sound workflow net out of an unsafe workflow net by introducing the notion of a layered petri net. (This might be already covered by object centric petri nets :( )
- Singleton pruning: Singletons are activities that only occurrs max. once in every case. (Define it formally in your paper) After the log enhancement, we can safely remove these activities from the set of enabled activities after its occurrence. This is a simple optimization step that can be done after the log enhancement step.
- Floating pruning: Floating activities are activities that are correlated with neither the control flow nor the data flow of the process. The basic assumptions of a process model is that every activity is enabled by the occurence of one or more previous activitie and/or the value of one or more data attribute values. (Take example of the *Sepsis* log) Activities such as *Leucocytes*, *LacticAcid* or *CRP* can be fired independently of the main control flow. These activities can therefore be added to every set of enabled activities.
- What can be the criteria for an activity to be classified as a floating activity?