0.0.1 Process Logs, Cases and Instance Sequences

We start by formalising the event log and its elements. We use a medical process as an example to provide a better semantic understanding. An event log is denoted as L. Here, L could be as database which logs the medical histories of all patients in a hospital.

We assume the database to log all interactions, be it therapeutic or diagnostic and store them as an event with a unique identifier. Let $\mathcal{E}$ be the universe of these event identifiers and $E \subseteq \mathcal{E}$ a set of events. The set E could consist, for instance, of a patients first session with a medical professional, then a diagnostic scan, followed by therapie sessions, surgery and more.

All of these interactions with one patient make up a case, which has a unique identifier, too. Let C be a set of case identifiers and $\pi_\sigma : E \mapsto C$ a surjective function that links every element in E to a case $c \in C$ in which c signifies a specific case. The function allows us to associate every event within the database to a single patient. The function's surjective property ensures for each case there exists at least one event.

For a set of events $E \subseteq \mathcal{E}$, we use a shorthand s^c being a particular sequence $s^c = \langle e_1, e_2, \ldots, e_t \rangle$ with c as case identifier and a length of t. Each s is an element of the process log $s \in L$. To understand the difference between c and s, we can say, that c is the ID for the case of patient X. Then, s^c is all the interactions that the database has logged for patient X.

These events are ordered in the sequence, in which they occured for patient X. Therefore, let $\mathcal{T}$ be the time domain and $\pi_t : E \mapsto \mathcal{T}$ a non-surjective linking function which strictly orders a set of events. In other words, every event in the database maps to one point in time. If the database logs every event on a daily basis, then all possible dates in history constitute $\mathcal{T}$. However, not every day has to be linked to a case as π_t is non-surjective.

Let $\mathcal{A}$ be a universe of attribute identifiers, in which each identifier maps to a set of attribute values $\overline{a}_i \in \mathcal{A}$. An attribute identifier describes everything the database might store for a patient, such as heart-rate or blood sugar level. If the database logs the heart-rate, then heart-rates of -42 beats-per-minute are not possible. Hence, $\overline{a}_i$ can per definition only map to positive integers.

Let $\overline{a}_i$ correspond to a set of possible attribute values by using a surjective mapping function $\pi_A : \mathcal{A} \mapsto A$. Then, each event e_t consists of a set $e_t = \{a_1 \in A_1, a_2 \in A_2, \ldots, a_I \in A_I\}$ with the size $I = |\mathcal{A}|$, in which each a_i refers to a value within its respective set of possible attribute values. In other words, every event consists of a set of values. If the event was recorded after a physio therapeutic session, then a_1 might be the specific degree to which you can move your ligaments and a_2 the medical professional's name. If the

event was recorded after a breast-cancer scan, the a_1, a_2 and a_3 might relate to the specific diameter, uniformity and aggressiveness values of the cancer. Conversely, we define a mapping from an attribute value to its respective attribute identifier $\pi_{\overline{a}} : A \mapsto \mathcal{A}$. Hence, we can map typically trace every attribute value back to it's attribute identifier.

The following part is not necessarily connected with *what* is stored within the database symbolically, but rather *how* it is represented in the database or during processing. We require a set of functions F to map every attribute value to a representation which can be processed by a machine. Let $\pi_d : A_i \mapsto \mathbb{N}$ be a surjective function, which determines the dimensionality of a_i and also F be a set of size I containing a representation function for every named attribute set. We denote each function $f_i \in F$ as a mapper to a vector space $f_i : a_i \mapsto \mathbb{R}_i^d$, in which d represents the dimensionality of an attribute value $d = \pi_d(A_i)$. For instance categorical variables will can map to a one-hot-encoded vector. Numerical values like heart-beat maight be recorded in a scalar form.

With these definitions, we denote any event $e_t \in s^c$ of a specific case c as a vector, which concatenates every attribute representation f_i as $\mathbf{e}_t^c = [f_1; f_2; \ldots; f_I]$. Therefore, $\mathbf{e}_t^c$ is embedded in a vector space of size D which is the sum of each individual attribute dimension $D = \sum_i \pi_d(A_i)$. Furthermore, if we refer to a specific named attribute set A_i as a name, we will use the shorthand $\overline{a}_i$. In other words, we concatenate all representations, whether they are scaler or vectors to one final vector representing the event.

Figure 1 shows a schematic representation of a log L, a case c and an event e.

0.0.2 State-Space Models

Generally speaking, every time-series can be represented as a state-space model[1]. Within this framework the system consists of *input states* for *subsequent states* and *subsequent outputs*. A mathematical form of such a system is shown in Equation 1.

$$\begin{aligned} \mathbf{z}_{t+1} &= h(t, \mathbf{z}_t, \mathbf{u}_t) \\ \mathbf{e}_t &= g(t, \mathbf{z}_t, \mathbf{u}_t) \\ \mathbf{z}_{t+1} &:= \frac{d}{dt}\mathbf{z}_t \end{aligned} \tag{1}$$

Here, $\mathbf{u}_t$ represents the input, $\mathbf{z}_t$ the state at time t. The function h maps t, $\mathbf{z}_t$ and $\mathbf{u}_t$ to the next state $\mathbf{z}_{t+1}$. The event $\mathbf{e}_t$ acts as an output computed

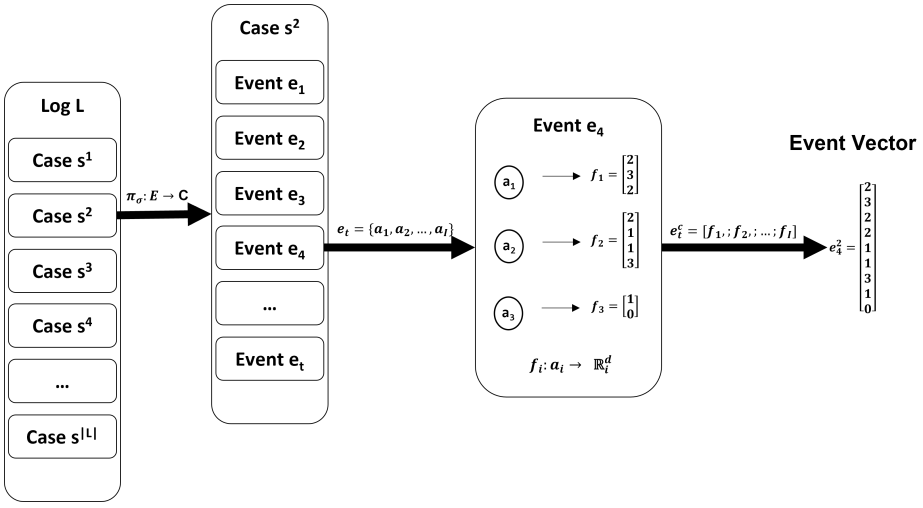

Figure 1: This figure shows the representation of a log L which contains anumber of cases s. Case s^2 contains a number of events e_t. Each events has attribute values a_i, which are mapped to vector spaces of varying dimensions. At last, all of the vectors are concatenated.

by function g which takes the same input as h. The variables $\mathbf{z}_t$, $\mathbf{u}_t$ and $\mathbf{e}_t$ are vectors with discrete or continuous features. The distinction of $\mathbf{z}_{t+1}$ and $\mathbf{e}_t$ decouples *hidden*[1] states, from *observable* system outputs. Figure 2 shows a graphical representation of these equations.

The body of literature for state-space models is too vast to discuss them in detail[2]. However, for process mining, we can use this representation to discuss the necessary assumptions for process mining. In line with the process-definition in **??**, we can understand the event log as a collection of the observable outputs of a state-space model. The state of the process is hidden as the *true* process which generated the data cannot be observed as well. The time t is a step within the process. Hence, we will treat t as a discrete scalar value to denote discrete sequential time steps. Hence, if we have $\sigma = \{a, b, b, c\}$, then t, describes the index of each element in σ. The input $\mathbf{u}_t$ represents all context information of the process. Here, $\mathbf{u}_t$ subsumes observable information such as the starting point and process instance-related features. The functions h and g determine the transition of a process' state to another state and its output over time. Note, that this formulation disregards any effects of future timesteps on the current timestep. Meaning, that the state

[1]A state does not have to be hidden. Especially, if we know the process and the transition rules. However, those are often inaccessible if we only use log data. Instead, many techniques try to approximate the hidden state given the data instead.

[2]For an introduction to state-space models see: XXX

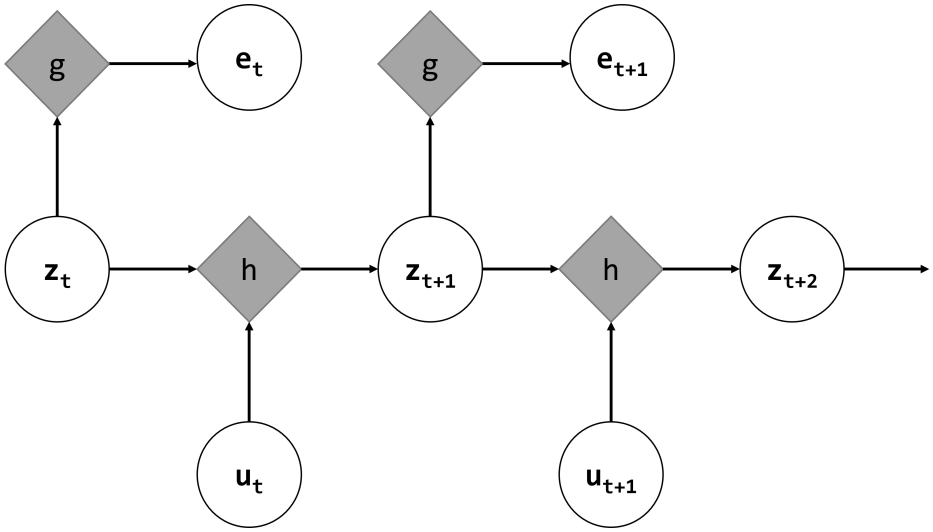

Figure 2: This figure shows a simplified graphical representation of a state-space model. Each arrow represents the flow of information.

transitions are causal and therefore, ignorant of the future. As we establish in **??**, we can assume that a process is a discrete sequence, whose transitions are time-variant. In this framework, we try to identify the parameters of the functions h and g. Knowing the functions, it becomes simple to infer viable counterfactuals. However, the function parameters are often unknown and therefore, we require probablistic approaches.

We can formulate Equation 1 probablistically as shown in Equation 2.

$$\mathbb{E}[p\left(z_{t+1} \mid t, z_{1:T}, u_{1:T}, x_{1:T}, \theta_h\right)] = \int z_{t+1} \cdot p\left(z_{t+1} \mid t, z_{1:T}, u_{1:T}, x_{1:T}, \theta_h\right) \quad (2)$$

$$\mathbb{E}[p\left(x_t \mid t, z_{1:T}, u_{1:T}, \theta_g\right)] = \int x_t \cdot p\left(x_t \mid t, z_{1:T}, u_{1:T}, \theta_g\right)$$

Note, that h and g are substitued with probability density functions parametrized with θ_h and θ_g. T signifies the full sequence including future timesteps. Both expectations are intractable as they require integrating over n-dimensional vectors. To solve the intractability, we characterize the system as a *Hidden Markov Process* and Probalistic Graphical Model (PGM). This framework allows us to leverage simplifying assumptions such as the independece from future values and *d-seperation*.

These characteristics change the probabilities in Equation 2 to Equation 3:

$$p\left(z_{t+1} \mid z_{1:t}, u_{1:t}, \theta_h\right) = \prod_{1}^{t} p\left(z_t \mid z_{1:t}, u_t, \theta_h\right) \tag{3}$$

$$p\left(x_t \mid z_{1:t}, \theta_g\right) = \prod_{1}^{t} p\left(x_{t-1} \mid z_{1:t}, \theta_f\right) \tag{4}$$

For $p\left(z_{t+1} \mid t, z_{1:T}, u_{1:T}, x_{1:T}, \theta_h\right)$, we ignore future timesteps, as T changes into t. *d-seperation* allows us to ignore all $\mathbf{e}_t$ of previous timesteps. The graphical form also decomposes the probability into a product of probabilities that each depend on all previous states and its current inputs. Previous $\mathbf{e}_t$ are ignored due to *d-seperation*. $p\left(x_t \mid t, z_{1:T}, u_{1:T}, \theta_g\right)$ only depends on its current state, which is in line with Hidden Markov Models (HMMs). Note, that we deliberately not assume a *strong Markov Property*, as the Deep Learning-Framework allows us to take all previous states into account. The *strong Markov Property* would assume that only the previous state suffices. At last, we assume that we do not model automatic or any other process whose state changes without a change in the input or previous states. Hence, we remove the dependency on the independet t variable. Only the previous states $z_{1:T}$ and the input information $\mathbf{u}_t$ remain time-dependent.

In this probablistic setting, the generation of counterfactuals, amounts to drawing samples from the likelihood of Equation 3. We then use the samples to reconstruct the most-likely a counterfactual $e^*_{1:t}$. Hence, our goal is to maximize both likelihoods.