Generating Synthetic Event Datasets for Tuning Root Cause Analysis Algorithms

D. Gueorguiev 7/19/23

Notation

G - directed graph
$\mathcal{V}(G)$ - the vertex set of the directed graph G
$\mathcal{A}(G)$ – the arc set of the directed graph G
E_x – event type for which we would like to do root cause analysis
$\mathcal{Y}(E_x)$ – set of event types associated with the event E_x subject to analysis
$\mathcal{E} \triangleq E_x \cup \mathcal{Y}(E_x)$ – set of all event types
$\mathcal{D}_\sigma[\mathcal{E}]$ - the set of all instances of associated events of types in $\mathcal{E}$
$\mathfrak{S}_c[\mathcal{E}, \mathcal{D}_\sigma]$ – the set of all constraints applied to event types in $\mathcal{E}$ and their instances $\mathcal{D}_\sigma[\mathcal{E}]$

The Problem of Generating Synthetic Data

Let us denote by $\mathcal{Y}$ the set of the event types which are relevant in root cause analysis of specific event type E_x. That is, the event types $E_1, E_2, \dots, E_k \in \mathcal{Y}(E_x)$ and the event type E_x will form causal pairs for which we want to calculate causal significance factor and construct Directed Causal Graph (DCG).
We will assume that we can have multiple instances of each event type $E_i|_{\tau_1}, E_i|_{\tau_2}, \dots, E_i|_{\tau_j}$. Here $\tau_1, \tau_2, \dots, \tau_j$ denote different sets of arguments for the same event type E_i.

We will consider the following constraint types which can be imposed on the event types. We may impose *directly follows* ($\prec$) constraint and *reachable from* ($<$) constraint to a subset of event types. Generalization of the *reachable from* constraint is *reachable from in at least* $\boldsymbol{a}$ *steps* ($<^{\geq a}$) and *reachable from in at most* $\boldsymbol{a}$ *steps* ($<^{\leq a}$). Another relevant constraint is the *multiplicity type* constraint with possible multiplicity types: *max children count* $\mu_\pi(E_i, E_j)\big|_\mathcal{E}$, *min children count* $\varepsilon_\pi(E_i, E_j)\big|_\mathcal{E}$, *max total count* $\mu_o(E_i)|_{\mathcal{D}_\sigma[\mathcal{E}]}$ and *minimum total count* $\varepsilon_o(E_i)|_{\mathcal{D}_\sigma[\mathcal{E}]}$ for an event type E_i within the dataset $\mathcal{D}_\sigma[\mathcal{E}]$ of *associated events*[1][2][3][4][5].

Note: The most general set of constraints which deal with non-deterministic conditions can be expressed using *Probabilistic Temporal Logic* (PTL) (for details see Appendix). For instance, $\mathrm{E}_a <_{\geq p}^{\geq r, \leq s} \mathrm{E}_b$ denotes that event E_b is reachable from event E_a with probability at least p after at least r time steps and at most s time steps.

Let us denote the set of all constraints imposed on even types in $\mathcal{E}$ and event instances in $\mathcal{D}_\sigma[\mathcal{E}]$ with $\mathfrak{S}_c[\mathcal{E}, \mathcal{D}_\sigma]$. We would like to construct a sequence of events from the specified type set $\mathcal{E}$ obeying the set of constraints $\mathfrak{S}_c[\mathcal{E}, \mathcal{D}_\sigma]$.

How to do that?
Idea: We can represent the events in $\mathcal{E}$ by a *Kripke* structure which will be subject to the set of constraints $\mathfrak{S}_c[\mathcal{E}, \mathcal{D}_\sigma]$

Let us build an example Kripke structure for our Fulfillment Decisions Root Cause Analysis problem discussed in (Gueorguiev, 2023).

---

[1] μ is from **μ**έγιστο (Greek for *maximum*)
[2] ε is from **ε**λάχιστο (Greek for *minimum*)
[3] The subscript π is from **π**αιδί (Greek for *child*)
[4] The subscript o is from **ο**λικός (Greek for *overall*)
[5] The subscript σ is from **σ**υνεταιρισμός (Greek for *association*)

Representing Fulfillment Event Dataset with *Kripke* Structure

Let us consider an event dataset represented by *timestamp-marked stream* of *event instances*:
$e_1, e_2, e_3, \dots, e_k, \dots, e_n, \dots$
Here each event instance e_k is an instance of some event type in $\mathcal{E}$ created at time t_i, for some set of arguments $\tau_{j,k}$ which belong to the value space $\mathcal{V}_j$ of all possible argument values of E_j (see paragraph *Events* of (Gueorguiev, 2023) for details). Here the index $k = 1,2,..$ represents the k-th appearance of the event type E_j in the timestamp-marked event stream. That is:

$$e_i = E_j\big|_{t_i,\tau_{j,k}}, \tau_{j,k} \in \mathcal{V}_j, t_i \in \mathbb{R} \quad (1)$$

From this timestamp-marked stream when t_1 and t_n are given we can always construct a *Kripke* structure $M|_{[t_1,t_n]}$. The algorithm for constructing such structure $M|_{[t_1,t_n]}$ is discussed below:

We will construct a sequence of Directed Follow Graph Instances (DFGI) $G_1, G_2, \dots, G_m$ where each DFGI G_i will contain a pair of special events – *starting event* E_s and *ending event* E_e.
Let us denote all instances of E_s with $e_k^{(s)}$ where the index $k = 1,2,..$ represents the k-th appearance of the event type E_j in the timestamp-marked event stream. Similarly, we denote all instances of E_e with $e_k^{(e)}$. We can visualize the timestamp-marked stream using this new notation as:

$$e_1, \dots, e_{k_1}, e_1^{(s)}, e_{k_1+1}, \dots, e_{k_1+l_1}, e_1^{(e)}, e_{k_1+l_1+1}, \dots, e_{k_1+k_2+l_1}, e_2^{(s)}, e_{k_1+k_2+l_1+1}, \dots, e_{k_1+k_2+l_1+l_2}, e_2^{(e)}, e_{k_1+k_2+l_1+l_2+1}, \dots$$

We can represent the structure of the timestamp-marked stream as a composition of the following finite sequences:

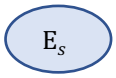
$$K_1, S_1E_1, K_{12}, S_2E_2, K_{123}, S_3E_3, \dots, K_{12..m}, S_mE_m, \dots$$

K_1 denotes a finite sequence of events which does not contain an instance of E_s or E_e.

S_mE_m denotes a finite sequence of events which is starting with an instance of E_s and ending with an instance of E_e where all events in the sequence between the first and the last event are not instances of E_s and E_e.

$K_{12..m}$ $(m > 1)$ denotes the finite sequence of events a) which does not contain an instance of E_s or E_e and is b) between two sequences, $S_{m-1}E_{m-1}$ and S_mE_m.

We label every state node of the Kripke structure with a tuple corresponding to the argument values of an event instance in the event stream which the structure will represent. Depending on the desired granularity level we can have more elaborate or less detailed event arguments. For example, let us consider the starting event E_s to be "*order is received*". Clearly, we can have different number of arguments describing each instance of the event "*order is received*". For instance, we can have each received order described only with the number of bundles in the order. That is, each instance of event "*order is received*" which has a single bundle will be undistinguishable from any other instance of the same event with single bundle. If we want to have more granular information when order is received, we can specify a SKU which is part of the order and the units requested for that SKU.



$b = 1, s = '0001', n_s = 1$

Specifying more granular parameter space for the event node will be reflected in a more complex (more branching) Kripke structure corresponding to the event dataset. Following the notation introduced in section

Events of (Gueorguiev, 2023) we denote the parameter space of the i-th event with $\mathcal{P}_i$. Let us denote with $\mathfrak{P}$ (*fraktur P*) the set of all parameter spaces corresponding to all events contained in $\mathcal{E}$.

Merging of the DFGI graph instances into a Kripke structure

Let us denote the set of all DFGIs with $\mathcal{G} \equiv \{G_1, G_2, \dots, G_m\}$. Each event sequence $S_i E_i, i = 1..m$ will result in a DFGI instance G_i.

Enumeration of parameter spaces by (structural) granularity

Let us denote all possible parameter spaces for i-th event with $\mathcal{P}_{i,*}$. On the example above $\mathcal{P}_{s,*} = [\{b\}, \{b, s\}, \{b, s, n_s\}]$. Having a single event parameter which specifies the number of bundles in the received order, having second parameter specifying the SKU number, having a third parameter specifying the number of units for the specified SKU are all viable parameter spaces. Notice that the possible parameter spaces can be enumerated and indexed. We can impose strong partial order relation $\lessdot$ to compare parameter spaces in terms of degree of granularity. In our example, clearly, $\{b\} \lessdot \{b, s\} \lessdot \{b, s, n_s\}$. Note that if $\mathcal{P}_{l,*}$ represents the enumerated by granularity parameter spaces of some event $E_l \in \mathcal{E}$ then $\mathcal{P}_{l,i} \lessdot \mathcal{P}_{l,j}$ can be true *iff* $i > j$. Note we can enumerate and index the set of the parameter spaces $\mathfrak{P}$ of all events in $\mathcal{E}$ by degree of granularity as well. With $\mathfrak{P}_* = \{\mathfrak{P}_1, \mathfrak{P}_2, \dots, \}$ will be denoted the parameter spaces for the events in $\mathcal{E}$ enumerated by granularity. For any two elements of $\mathfrak{P}_*$, $\mathfrak{P}_i \lessdot \mathfrak{P}_j$ can be true *iff* $i > j$.

Algorithm:
Step 1: Construct the set $\mathcal{G}$ of Directed Follow Graph Instances (DFGI) $G_i, i = 1, \dots, m$ each corresponding to a pair of (E_s, E_e) instances.
Step 2: Construct an incomplete Kripke structure from $\mathcal{G}$.
Step 3:
Step 4:

Bibliography

Gueorguiev, D. (2023). *Root Cause Analysis For Fulfillment Decisions.* Boston, MA.

Hans Hansson, B. J. (1994). *A Logic for Reasoning about Time and Reliability.* Kista, Sweden: Swedish Institute of Computer Science.

The report "*Root Cause Analysis For Fulfillment Decisions*" can be found here.

Appendix: Probabilistic Temporal Logic

A detailed and thorough survey by Clarke et al on the most relevant logic systems for model verification can be found here: Clarke, E. M., Schlingloff, B. H. (2001). Model Checking

As quick intro into Temporal Logic can serve Clarke, Emerson, and Sistla's paper: Clarke, E.M., Emerson, E.A., Sistla, A.P. (1983). Automatic Verification Finite State Concurrent Systems Using Temporal Logic Specifications: A Practical Approach

An excellent tutorial for Probabilistic Temporal Logic (PTL) is the Hanssen and Jonsson's paper: Hansson, H., Jonsson, B. (1994). A Logic about Reasoning about Time and Reliability

A refresher on first order logic which is the fundament of the logic systems for model verification can be found here: First-Order Logic, Open Logic Project

Examples of PTL expressions

$\mathrm{E}_a <_{\geq p}^{\geq r, \leq s} \mathrm{E}_b$ denotes that event E_b is reachable from event E_a with probability at least p after at least r time steps and at most s time steps. This operator is also known as the *quantified leads-to* operator discussed in (Hans Hansson, 1994).