Statistical Log Differencing

Lingfeng Bao*, Nimrod Busany†, David Lo‡, and Shahar Maoz†
*Zhejiang University City College, China
†Tel Aviv University, Israel
‡Singapore Management University, Singapore

***Abstract*—Recent works have considered the problem of log differencing: given two or more system's execution logs, output a model of their differences. Log differencing has potential applications in software evolution, testing, and security.**

In this paper we present statistical log differencing, which accounts for frequencies of behaviors found in the logs. We present two algorithms, s2KDiff for differencing two logs, and snKDiff, for differencing of many logs at once, both presenting their results over a single inferred model. A unique aspect of our algorithms is their use of statistical hypothesis testing: we let the engineer control the sensitivity of the analysis by setting the target distance between probabilities and the statistical significance value, and report only (and all) the statistically significant differences.

Our evaluation shows the effectiveness of our work in terms of soundness, completeness, and performance. It also demonstrates its effectiveness compared to previous work via a user-study and its potential applications via a case study using real-world logs.

I. Introduction

Many works present different kinds of log analyses, which take a system's execution log as input and output a model, e.g., a state machine, a set of scenarios, or a set of invariants, which provide data and insights about the behavior of the system that produced the log (see, e.g., [4], [14], [22]). One kind of log analysis is differencing, i.e., comparing logs in order to infer and represent differences between two or more sets of executions [3], [16]. Log differencing has various applications in software engineering. One example application is in the context of testing, e.g., comparing logs produced in the lab vs. ones produced in the field, to improve test design and execution. Another example application is in the context of security, e.g., comparing logs produced by a benign app with ones produced by a similar app, suspected to be infected with malicious code. Additional example applications include comprehension in the context of software evolution.

In recent work [3], we have presented a log differencing approach based on finite-state models and a notion of k-differences, sequences of k events that appear in one log and not the other. The work presented 2KDiff, an algorithm that compares two logs, and nKDiff, an algorithm that compares several logs at once. In both, the results are presented succinctly on top of a single finite-state model, with relevant transitions highlighted. An important strength of these two algorithms is that they are sound and complete: all reported differences are indeed differences, and all differences are reported.

One clear weakness of the 2KDiff and nKDiff algorithms, however, is that they abstract away the frequencies of the different behaviors in the logs. This results in two limitations that affect usefulness. First, logs that include the same set of behaviors albeit at very different frequencies are reported as equivalent, with no differences found. Second, when several differences are reported, all are presented as equally important.

In this paper we present statistical log differencing, which accounts for frequencies of behaviors. Following our previous work [3], we present s2KDiff, an algorithm to compare two logs, and snKDiff, an algorithm to compare several logs at once, both, importantly, with an additional statistical dimension as follows.

Almost any two logs coming from different sets of runs of the same system will not be identical in terms of the frequencies of different behaviors. However, subtle differences in frequencies may be meaningless. Moreover, large differences in frequencies that are based only on a small set of traces may not actually represent true differences. Where should the line be drawn between interesting and uninteresting differences? Our answer is based on statistical hypothesis testing. We let the engineer control the sensitivity of the analysis by setting the target distance between (transition) probabilities, e.g., $d = 0.1$, and the statistical significance value, e.g., $\alpha = 0.05$. We then report only (and all) the statistically significant differences.

On top of the use of hypothesis testing, our approach has three additional important features. First, we order the differences by their statistical significance, and report them one by one, on demand, in this order. This helps engineers focus on the more significant differences first. Second, we compute the differences locally, at the level of transition probabilities, but report them within the larger context of the complete behavior, represented by a (k-Tails [6] based) finite-state model (and, when applicable, a concrete trace from the log). This helps engineers conceptualize the behavior in question in its correct context. Finally, when differencing many logs at once, we present a single compact model that includes log indices over the transitions. Logs that are statistically different are put into different groups. This facilitates the multiple log comparison, and allows engineers a quick identification of the differences and the logs where they appear.

We have implemented our algorithms, validated the correctness of the implementation, and evaluated it against logs generated from models taken from several sources. Our evaluation shows that the statistical analysis is effective in terms of providing the expected statistical guarantees and in terms of its performance. We report a user-study of 20 participants showing the effectiveness of the algorithms in identifying behavioral differences in comparison to the baselines. We further report on a case study, using real-world logs from the work of Ghezzi et al. [15], which demonstrates one potential use of statistical log differencing in practice.

II. Example

We use a small and simple example to demonstrate the ability of s2KDiff and snKDiff to reveal significant differences of behaviors between 2 logs and n logs resp.

Consider the CVS Client model from Lo and Khoo [21]. The model has 18 states and 28 transitions with an alphabet of 15 labels. For our example here, we manually created two copies of this model, each annotated with different probabilities on its transitions. We then generated 3 logs from each of the two models, and refer to them as `L1`, …, `L6`. Each log includes 100,000 traces. The 6 generated logs simulate the behavior of different clients. The engineer is interested to learn about significant differences in behavior profiles between the 6 clients.

For a high-level view of the differences across the logs, the engineer runs snKDiff. Figure 1a shows the output of snKDiff on the 6 logs, running with k=1, a distance d=0.3, and a significance value of α=0.05. It consists of a finite-state model that captures all the behaviors found in the logs, and a csv file with a summary of the comparisons between the logs.

In the output, each state is followed by a single event label that appears in at least one of the logs. A transition between two states s_i, s_j, indicates that the two corresponding events appeared consecutively in at least one trace in at least one of the logs. For each transition, and for each log, snKDiff computes the probability of taking the transition. Then, it conducts a set of statistical tests and considers a transition to be differing if there exists at least one pair of logs with a transition probability difference above d and a significance of α.

Each such transition is highlighted and labeled with a grouping of the logs' indices according to their transition probabilities. Logs that belong to different groups are statistically different. The transition also includes the probabilities of making the transition in each of the log groups. The width of the transition highlights the significance of the difference in comparison to the other statistically significant differences. In Fig. 1a, for example, the most significant differences are from state 10, which is followed by the event `rnm` (rename), to states 3 and 4, which are followed by the `strfl` (storefile), `logout` labels resp. Consider the transition from state 10 to state 4. It shows that the probability of taking the transition, equals 87% in `[L1, L2, L3]`, and 43% in `[L4, L5, L6]` resp. The transition highlights the major probability difference in obtaining the sequence of events $\langle$`rnm`, `logout`$\rangle$ between the two groups of logs. This demonstrates the ability of snKDiff to reveal significant differences between sets of logs.

The engineer can now continue to "drill down" to inspect differences between two selected logs, using s2KDiff. Figure 1b shows the output of s2KDiff on logs `L1` and `L4`, again with k=1, $d = 0.3$, and $\alpha = 0.05$. The output of s2KDiff consists of two finite-state models that capture all the behaviors found in `L1` and `L4` resp. We focus on the model of `L1`. For each transition, s2KDiff computes the probability of taking the transition in `L1` and `L4`. Then, it conducts a statistical proportion test and considers the transition to be differing iff its probability difference is above d with a significance value of α.

The tool shows one differing transition at a time. For each, it finds and highlights the trace in `L1` that includes it, and has the greatest probability difference in being accepted by the two models. For example, Fig. 1b shows the model of `L1`. It highlights the significant difference from state 8 to state 3, which corresponds to the 2-sequence $\langle$`setfl` (setfiletype), `strfl` (storefile)$\rangle$ (red bold transition). The transition label shows the transition probabilities 0.53 and 0.06 in each of the models, and the statistical test's p-value. Finally, s2KDiff highlights the accepting path of the selected trace (dashed red lines): $\langle$`init` (initialize), `login`, `setfl`, `strfl`, `apndfl` (appendfile), `logout`, `dscon` (disconnect)$\rangle$.

III. Preliminaries

We present preliminary definitions that we use later in the paper. We start with traces and finite-state machines, and continue with an overview of the k-Tails algorithm. We recall earlier log differencing algorithms and conclude with several well-known statistical definitions that we use in our work.

A. Basic Definitions

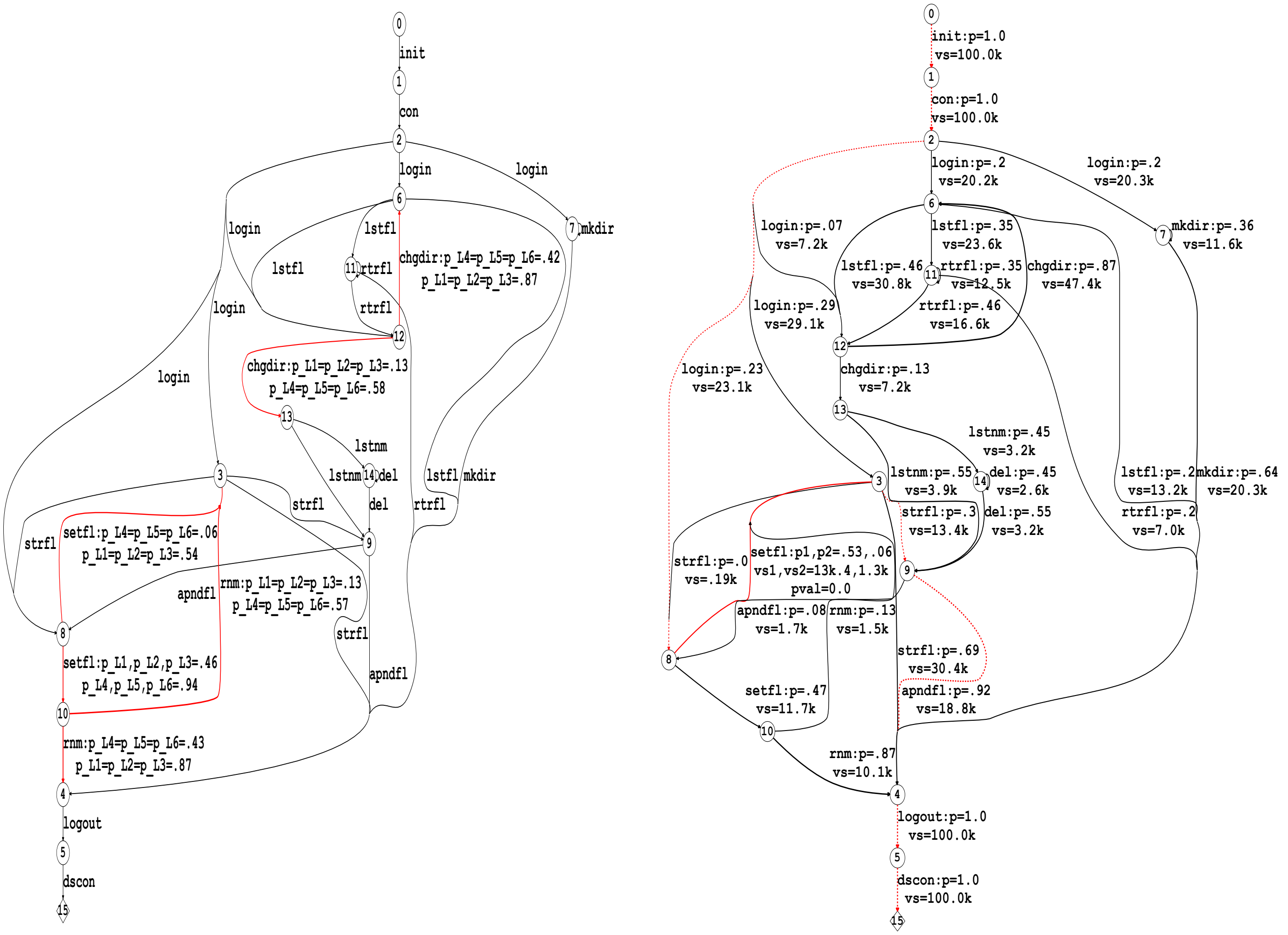
A trace over an alphabet of events Σ is a finite word $tr = \langle e_1, e_2, \ldots, e_m \rangle$ where $\forall e_i \in tr.\, e_i \in \Sigma$. For $j \geq 1$ we use $tr(j)$ to denote the jth element in tr. We use $|tr|$ to denote the length of tr. For a positive integer k, a k-sequence is a consecutive sequence of k (or less) events, denoted by seq_k. $\Sigma_{\leq k}$ is the set of all k-sequences over Σ. A log L over an alphabet Σ is a set of traces $L = \{tr_1, \ldots, tr_n\}$.

Definition 1 (Finite-State Machine (FSM))**.** *A finite-state machine (FSM) is a structure $M = \langle Q, Q_i, Q_s, \Sigma, \delta \rangle$ where Q is a set of states; $Q_i \subseteq Q$ is a set of initial states; $Q_s \subseteq Q$ is a set of accepting states; Σ is an alphabet; and $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ is a transition relation, where $\mathcal{P}(Q)$ is the power set of the set of states Q.*

We use subscript notation to refer to the elements of the FSM. For example, δ_M refers to the transition relation of M. Let M be an FSM over an alphabet Σ. We use $\mathcal{L}(M) \subseteq \Sigma^*$ to denote the set of all words accepted by M.

Definition 2 (Probabilistic Finite-State Machine (PFSM) [35])**.** *A probabilistic finite-state machine is a tuple $M = \langle Q, \Sigma, \delta, \mathcal{I}, \mathcal{F}, \mathcal{P} \rangle$ where Q, Σ, δ are defined similar to an FSM, $\mathcal{I}/\mathcal{F} : Q \to [0,1]$ are initial and terminal state probabilities resp., and $\mathcal{P} : Q \times \Sigma \times Q \to [0,1]$ are transition probabilities. Further, $\Sigma_{q \in Q} \mathcal{I}(q) = 1 \wedge \forall q \in Q, \mathcal{F}(q) + \Sigma_{a \in \Sigma, q' \in Q} \mathcal{P}(q, a, q') = 1$.*

This definition requires that the probability of starting at any of the states in Q equals 100%, and that for any state $q \in Q$, the probability of transitioning out or terminating equals 100%.



(a) Output of snKDiff comparing logs `L1`, ..., `L6` (see Sect. II), with k=1, d=0.3, α=0.05. Red transitions correspond to k-differences, and are labeled with log indices, s.t. logs in different groups differ statistically.

(b) Output of s2KDiff comparing `L1` and `L4` (see Sect. II), with k=1, d=0.3, α=0.05. Model for log `L1`. The red, bold transition highlights a k-difference between the two logs. The dashed line corresponds to a selected trace from `L1` that includes the k-difference.

Fig. 1: Example outputs of snKDiff and s2KDiff

B. k-Tails

k-Tails [6] is a classic model inference algorithm, which has been presented in several variants and implemented in many works, e.g., [5], [10], [24], [25], [27]. k-Tails takes a log and a parameter k as input. It starts by representing the log as an FSM M_{lin} composed of linear sub-FSMs, one per trace, which are joined by adding a single initial state q_{init} transitioning to the start of each trace via a unique α label, and a single terminal state q_{acc} to which all traces transition to at the end via a unique ω label. Notice that the language of M_{lin} equals the set of traces in the log, given that each trace is encapsulated by α and ω events. We refer to this version of the log as the encapsulated version, denoted L_{en}. k-Tails iteratively merges states in the M_{lin} FSM: Two states are merged iff they are k-equivalent, i.e., if their future of length k or less, is identical. When no two remaining states are k-equivalent, the algorithm terminates and outputs the resulting FSM, called a k-FSM.

More formally, we define a function $future : Q_{M_{lin}} \to \mathcal{P}(\Sigma_{\leq k})$, mapping states in M_{lin} to k-sequences. The k-equivalence relation induces a partition of the states of the initial FSM M_{lin} into equivalence classes $E = \{e_1, e_2, \ldots, e_m\}$, where each of the equivalence classes in E is uniquely defined by its future sequences of length k or less: two states $s_1, s_2 \in e_i$ iff $future(s_1) = future(s_2)$. When lifted from $Q_{M_{lin}}$ to E, the function *future* becomes the injective function $id : E \to \mathcal{P}(\Sigma_{\leq k})$. For all $s \in e_i$, $future(s) = id(e_i)$.

Definition 3 (k-FSM)**.** *k-FSM, the FSM computed by k-Tails for a log L and a positive integer k, is an FSM $M_L = \langle Q, Q_i, Q_s, \Sigma, \delta \rangle$ where $Q = E$, the set of equivalence classes defined above; Σ is the alphabet of the log L; $\forall e \in E. a \in \Sigma. \delta(e, a) = \bigcup \{e' | \exists s, s' \in M_{lin}. s \in e \wedge s' \in e' \wedge s' \in \delta_{M_{lin}}(s, a)\}$; $Q_i = \{q_{init}\}$ is an artificial initial state; and $Q_s = \{q_{acc}\}$ is an artificial terminal state.*

For a given k-FSM M_L, generated by running k-Tails on log L, we use $\mathcal{L}(M_L)$ to denote the set of all words accepted by M_L. Among other properties, the correctness of the k-Tails algorithm implies that M_L may over approximate the set of traces in L, but may not under approximate it, i.e., $L \subseteq \mathcal{L}(M_L)$. Consequently, every k-sequence included in any trace in L, is part of at least one accepting word of M_L. Additional useful properties of the k-FSM are that all its states are reachable from the initial state q_{init}, and that the accepting state q_{acc} is reachable from all states.

Further, since we use a variant of k-Tails that starts from M_{lin}, and since each state in M_{lin} is followed by a single future (apart from the dummy initial state), then all states in the k-FSM are followed by single k-future. As a corollary, any sequence of k events that appears in the log corresponds to a unique k-FSM state, and any k+1 sequence that appears in the log corresponds to a unique k-FSM transition.

C. Earlier Log Differencing Algorithms

We recall two existing techniques for log differencing, 2KDiff and nKDiff [3], which we extend in this work.

2KDiff. 2KDiff is a sound and complete extension of k-Tails, for comparing two logs. Given a positive integer k, 2KDiff compares two logs by focusing on k-differences, i.e., k-sequences that appear in one log but not the other, and present them in the context of a k-FSM model using highlighted traces from the logs. 2KDiff is sound and complete modulo the k-Tails abstraction. Any k-sequence that appears in one log and not the other is included in at least one highlighted trace on the k-FSM of the respective log, and any such highlighted trace contains at least one such k-sequence.

nKDiff. nKDiff is a sound and complete extension of k-Tails, used for comparing many logs. Roughly, given a set of n logs, $\{L_1, \ldots, L_n\}$, and a positive integer k, nKDiff computes the k-DiffLFSM, a single FSM whose transitions are labeled with subsets of log indices.

Definition 4 (k-DiffLFSM)**.** *For set of logs* $L = \{L_1, \ldots, L_n\}$ *and a positive integer* k*, a k-DiffLFSM* $M_{L_1 \ldots L_n}$ *is an LFSM* $\langle Q, Q_i, Q_s, \Sigma, I, \delta, label \rangle$ *where:* $Q = E$ *is the set of equivalence classes of states from the k-FSM* M_L*;* I *is the set of indices* $\{1 \ldots n\}$*;* Σ *is the union of the alphabets of the logs* L_1 *to* L_n*;* $\forall e, e' \in E, a \in \Sigma$*:* $label(e, a, e') = \{j | \exists s, s' \in M^j_{lin}$ *s.t.* $s' \in \delta_{M^j_{lin}}(s, a) \wedge future^j(s) = id(e) \wedge future^j(s') = id(e')\}$*;* $\forall e, e' \in E, a \in \Sigma$*:* $e' \in \delta(e, a)$ *iff* $label(e, a, e') \neq \emptyset$*;* $Q_i = \{q_{init}\}$ *is an artificial initial state; and* $Q_s = \{q_{acc}\}$ *is an artificial terminal state.*

nKDiff is sound and complete: its projection on any given index results in the k-FSM of the log with that index (soundness), and any behavior that appears in at least one of the logs is included in it (completeness). The k-DiffLFSM highlights all transitions that correspond to k-sequences that only appear in some of the logs and not in others.

D. Statistical Definitions

We recall well-known statistical definitions that we use in our work. We will use Bernoulli trials to model the transitions in the inferred PFSM. We will use the $\mathcal{Z}$-test to test whether the difference between probabilities of corresponding transitions is significant. We will use the χ^2-test to compare corresponding transitions on many logs, and test whether there exists at least one pair of logs with different transition probabilities.

Definition 5 (Bernoulli trials [2])**.** *Let* $x_1, x_2, \ldots, x_n$ *be* n *identical independent trials, where each trial has two possible outcomes, referred to as 'success' and 'failure'. Let* p *denote the probability of a success in a single trial, and* Y *denote the number of successes in* n *trials. Then, the probability of an outcome,* $Y = y$*, follows the Binomial distribution:* $P(y \in [0, \ldots, n]) = \frac{n!}{y!(n-y)!} p^y (1-p)^y$*.*

Let $X_1 = \{x_{1,1}, \ldots, x_{1,n_1}\}$, and $X_2 = \{x_{2,1}, \ldots, x_{2,n_2}\}$ denote two samples of Bernoulli trials of size n_1 and n_2 with m_1 and m_2 successes, and let $\hat{p_1}$ and $\hat{p_2}$ denote the sample proportions of success (i.e., $\hat{p_i} = \frac{m_i}{n_i}$). Testing whether the difference between the proportions of two series of Bernoulli trials is smaller than a constant can be done using hypothesis testing. The $\mathcal{Z}$-test is a popular test for this purpose.

Definition 6 ($\mathcal{Z}$-test: Large-Sample Hypothesis Tests for Two Population Proportions, Using Independent Samples [38])**.** *The* $\mathcal{Z}$*-test is a hypothesis-testing procedure for comparing two population proportions.*

The null hypothesis of the test is $H_0 = p_1 - p_2 < \mathrm{d}$*, and the alternative hypothesis is* $H_a = p_1 - p_2 > \mathrm{d}$*. The value of the* $\mathcal{Z}$*-test is* $\mathcal{Z} = (\hat{p_1} - \hat{p_2} - \mathrm{d}) / \sqrt{\frac{\hat{p_1}(1-\hat{p_1})}{n_1} + \frac{\hat{p_2}(1-\hat{p_2})}{n_2}}$*.*

Using the statistic above, we calculate the critical value $\mathcal{Z}_\alpha$*. If the value of the statistics is larger than the critical value,* $\mathcal{Z} > \mathcal{Z}_\alpha$*, then we reject the null hypothesis* H_0*. The probability that we correctly rejected the null hypothesis is defined by* $p_{val} = 1 - Probability(z > \mathcal{Z})$*, a.k.a p-value.*

Definition 7 (Chi-Square Independence Test [38])**.** *The Chi-Square Independence Test (*χ^2*-test) is used to decide whether two variables,* v_1 *and* v_2*, are associated. The null hypothesis is that the two variables are not associated. The* χ^2*-test compares the observed frequencies with the frequencies that we would expect under the null hypothesis of non-association. The test statistic is* $x = \frac{\sum(O-E)}{E}$*, where* O *represents the observed frequency and* E *the expected frequency. To compute the likelihood of the statistics, i.e.,* $P(\chi^2 > x)$*, the* χ^2*-test requires the degree of freedom as input. Let* R*,* C *denote the number of values that* v_1*,* v_2 *take, then the degree of freedom of* χ^2 *is defined as* $(R-1)(C-1)$*.*

IV. Statistical Log Differencing

We now present the main contributions of our work, s2KDiff, for differencing two logs, and snKDiff, for differencing many logs. We give formal definitions and examples, but leave the proofs, complexity analysis, and additional remarks to [1].

A. From a k-FSM to a k-PFSM

Since s2KDiff and snKDiff identify differences between the transition probabilities in the k-PFSM models of the logs, we first explain how we compute these probabilities.

Let L and M denote a log and its corresponding k-FSM model. We show how the probabilistic terms of the k-PFSM (i.e., $\mathcal{I}$, $\mathcal{F}$, $\mathcal{P}$, Def. 2) are defined.

Let $t_M = (e, \sigma, e')$ denote a transition in M. Let $V(e)$ denote the number of visits made by traces in L to the state e_M, and by $V(t)$ the number of times that transition t is fired. Then, the maximum likelihood estimator (MLE) for the transition probability of t is given by $\frac{V(t)}{V(e)}$ [28]. We use the MLE formula to infer $\mathcal{P}$, the model transition probabilities.

Further, the k-FSM model has unique dedicated source and terminal states (by construction), hence we define $\mathcal{I} = \{Q_i \to 1, Q \setminus Q_i \to 0\}$, and $\mathcal{F} = \{Q_s \to 1, Q \setminus Q_s \to 0\}$. These fully specify the k-PFSM extension of the k-FSM.

Calculating states and transition counters.. Recall that each state in the k-FSM is an equivalence class of states in M_{lin}, and that each transition in M_{lin} corresponds to an event in a trace. Therefore, the number of visits (of traces) of a state $e \in M_Q$ is given by the number states (of M_{lin}) in e. The number of visits (of traces) of a transition $t_M = (e, \sigma, e')$, is given by the number of states (of M_{lin}) in e that are followed by states (of M_{lin}) in e'.

We compute the counters per state and per transition on the fly as the states of M_{lin} are merged. Note that since each state $e \in M$ is followed by a single k-sequence, a state (resp. transition) counter shows the number of times that each unique k-sequence (resp. (k+1)-sequence) appears in the log.

B. s2KDiff: Differencing Two Logs

At the heart of s2KDiff is a statistical engine that performs a series of statistical tests to identify statistically significant differences between the logs. To this end, we define the notion of a (k, d)-difference between two logs. A (k, d)-difference is a sequence of k+1 consecutive events that corresponds to a transition in the k-PFSMs, and has probability difference of at least d between the two models.

From here on, since each transition in the k-PSFM corresponds to a unique (k+1)-sequence, we use the terms interchangeably. Let us formalize the above.

Definition 8 (Corresponding transitions)**.** *Let L_1, L_2 denote two logs, and M_1, M_2 their corresponding k-PFSMs. Let t_{M_1}, t_{M_2} denote two transitions in M_1, M_2 resp. We say that t_{M_1} corresponds to t_{M_2} iff they correspond to an identical (k+1)-sequence.*

Definition 9 ((k, d)-difference[1])**.** *Let L_1, L_2 denote two logs, and M_1, M_2 their corresponding k-PFSMs. Consider seq_{k+1}, a sequence of length k+1. Assume that seq_{k+1} appears in both logs, and consider its corresponding transitions t_{M_1}, t_{M_2} in M_1, M_2. Let p_1, p_2 denote the transition probabilities of t_{M_1}, t_{M_2} resp. Given a distance $d \in [0, 1]$, we say that seq_{k+1} is a (k, d)-difference iff $|p_1 - p_2| > d$.*

Due to the probabilistic nature of the transitions' estimators, s2KDiff performs a statistical test to determine if a (k, d)-difference is statistically significant. Let us first formulate the probabilistic modeling of each transition.

Definition 10 (A transition as a Bernoulli trial)**.** *We model a transition $t = (e, \sigma, e')$ in the k-PFSM as a Bernoulli trial, $\mathcal{B}(p)$, where each time that e is visited it can fire t with probability p.*

Definition 11 (Trace visits as a series of Bernoulli trials)**.** *Let $t = (e, \sigma, e')$ denote a transition in the k-PFSM. We model each visit of a trace to e as a single trial in a series of Bernoulli trials. We consider a trial as a 'success', if t is fired, and 'failure' otherwise.*

To identify statistically significant (k, d)-differences, we use the above modeling, and preform the hypothesis test presented in Sect. III to compare the probabilities of corresponding transitions between the two models. Specifically, to compare corresponding transitions $t_{M_1} = (e_1, \sigma, e_1')$ and $t_{M_2} = (e_2, \sigma, e_2')$, we obtain the source state and transition counters: $V(e_1)$, $V(e_2)$, $V(t_1)$, $V(t_2)$ from the k-PFSM models. To use the formula of the $\mathcal{Z}$-test (Def. 6), we set $n_1 = V(q_1)$, $n_2 = V(q_2)$, $m_1 = V(t_1)$, $m_2 = V(t_2)$, a distance $d \in [0, 1]$ and a significance value $\alpha \in [0, 1]$. Let p_1, p_2 denote the transition probabilities in the two models. W.l.o.g. assume that $p_1 > p_2$. We execute the hypothesis test, setting the null hypothesis of the test to $H_0 : p_1 - p_2 < d$. We consider a transition as a (k, d)-difference, iff, the null hypothesis can be rejected, i.e., the test result has a p-value below α.

We now provide a high-level description of the s2KDiff algorithm. Given a positive integer k, a difference $d \in [0, 1]$, and a significance value $\alpha \in [0, 1]$, s2KDiff compares two logs by focusing on (k, d)-differences, and presents all the statistically significant ones in the context of the models, using traces from the logs.

The s2KDiff Algorithm. The algorithm gets two logs, k , d, and α as input. It computes the k-PFSM models of the two logs, enriched with the number of times that each state and transition are visited by the traces in the corresponding logs.

Then, it finds all pairs of corresponding transitions. For each transition, the algorithm computes the number of source state (n), and transition visits (m) in the two models and performs a $\mathcal{Z}$-test. If the p_{val} of the test is less than α, i.e., the transition corresponds the a (k, d)-difference, the algorithm stores it.

Then, for each (k, d)-difference, the algorithm searches for an evidence trace. To this end, the algorithm iterates over the traces in the logs, and selects the traces that include the (k, d)-difference. For each trace, it computes the acceptance probabilities in both models. After iterating all traces, it chooses the one with the maximal absolute acceptance probability difference.

Finally, it outputs the results of all statistical tests conducted (in a csv format), along with a list of (k, d)-differences and corresponding traces. It also outputs the graphs (in .dot format) with a highlighting of the most significant (k, d)-difference and its evidence trace.

[1] Note that a transition in a k-PFSM corresponds to a sequence of k+1 events in the log (see Sect. III), still, we refer to these as (k, d)-differences.

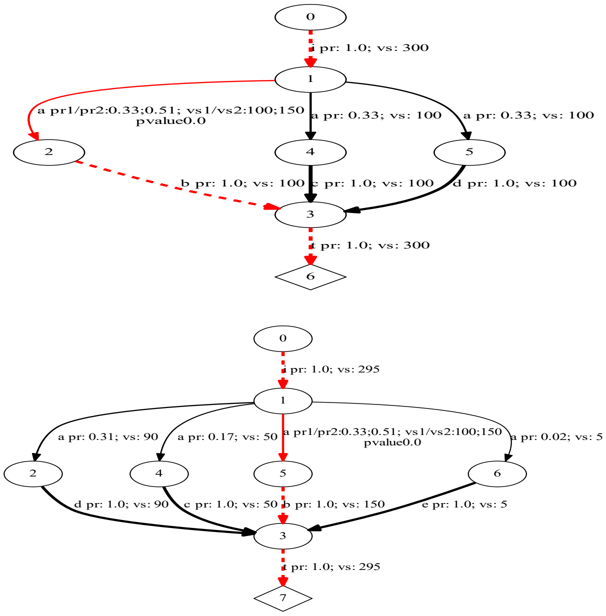


Fig. 2: The resulting models of s2KDiff when running with `L1` (top model), `L2` (bottom model), k=1, d=0.1, α=0.05. The logs are described in Example 1.

Example 1. *We demonstrate the algorithm with a simple example. Consider running s2KDiff with* k=1, d=0.1, α=0.05 *on* `L1`=$\{tr_1^{100}$:$\langle a,b\rangle, tr_2^{100}$:$\langle a,c\rangle, tr_3^{100}$:$\langle a,d\rangle\}$ *and* `L2`=$\{tr_1^{90}$:$\langle a,d\rangle, tr_2^{50}$:$\langle a,c\rangle, tr_3^{150}$:$\langle a,b\rangle, tr_4^{5}$:$\langle a,e\rangle\}$*, where the superscript shows the number of repetitions per trace. The algorithm computes the models based on the encapsulated logs and then finds all pairs corresponding transitions; e.g., it pairs transitions* t_{M_1}=$(1,a,2)$ *and* t_{M_2}=$(1,a,5)$*, which correspond to the k-sequence* $\langle a,b\rangle$*, and transitions* t_{M_1}=t_{M_2}=$(1,a,4)$*, which correspond to the k-sequence* $\langle a,c\rangle$*. Then, it iterates over 9 tuples of corresponding transitions, and performs a* $\mathcal{Z}$*-test per transition. It classifies the two pairs above as* (k,d)*-differences. For example, the first pair is highlighted in both models (Fig. 2). The parameters of the* $\mathcal{Z}$*-test are* n_1=$V_{M_1}(1)$=300, m_1=$V_{M_1}(1,a,2)$=100, n_2=$V_{M_2}(1)$=295, m_2=$V_{M_2}(1,a,5)$=150, *and the resulting* p_{val}=0.0297 *is well below* α*. Hence, s2KDiff considers it significant and highlights it. To find an evidence trace, it iterates over all traces. The trace* $\langle a,b\rangle$ *includes the k-sequence, and has an acceptance probability of* 33.3%, 51% *in the models of* `L1`, `L2` *resp. Since it is the only trace that includes the k-sequence, it is selected and highlighted.*

Note that the transition t_{M_2}=$(1,a,6)$*, does not have a corresponding transition in* M_1*. However, since its source state (k-future=*$\{a\}$*) appears in both models, it can be tested with* n_1=$V_{M_1}(1)$=300, m_1=0, n_2=$V_{M_2}(1)$=295, m_2=$V_{M_2}(1,a,6)$=5*. We omit the details of handling such cases.*

It is important to note that similar to 2KDiff, s2KDiff is sound and complete modulo the k-sequences abstraction.

Theorem 1 (s2KDiff Soundness and Completeness)**.** *Let* k *be a positive integer and let* L_1*,* L_2 *be two logs compared using s2KDiff with* M_1 *and* M_2 *their corresponding k-PFSM models. Then, any trace highlighted by s2KDiff over* M_1 *is a trace from* $L_1 \cup L_2$ *that includes at least one* (k,d)*-difference with respect to* L_2*. The same holds for* M_2*. Further, every* (k,d)*-difference between* L_1 *and* L_2 *is highlighted by at least one accepted trace in* M_1 *or* M_2*.*

C. snKDiff: Differencing Many Logs

We now present snKDiff, which extends nKDiff (see Sect. III): given a set of n logs $\{L_1,\ldots,L_n\}$, and a positive integer k, snKDiff computes a k-DiffLFSM model that maintains the original soundness and completeness properties. Unlike nKDiff, snKDiff accounts for the frequencies of k-sequences in the logs, and only highlights transitions that have different probabilities in the models of different logs. Importantly, only (and all) statistically significant differences are highlighted.

Recall our definition of a (k,d)-difference (Def. 9). snKDiff searches for (k,d)-differences by comparing the probabilities of corresponding transitions in the k-PFSMs of the n different logs (i.e., transitions with source and target states that map to identical k-futures). To this end, it conducts a series of statistical tests per transition.

A straightforward yet costly approach would solve snKDiff by performing s2KDiff on all transitions on all pairs of logs. This, however, will result in a $O(n^2)$ runs of s2KDiff per transition. Our approach is different. We start by employing a χ^2-test and then apply s2KDiff only to the transitions for which we have an indication of a possible difference. In our evaluation we show that this heuristics for improved performance does not increase the statistical error. This is expected because the χ^2-test is conservative, as it only checks for a possible difference, without accounting for the difference size.

Specifically, in Sect. III we presented the χ^2-test for testing if two variables are associated. We established that each transition can be viewed as a random variable of a Bernoulli trial. This is our first variable for the test. For the second variable, we consider all the realizations of a transition in all n logs. Then, we model the log id as a variable that characterizes the realizations. We conduct a χ^2-test to test if the variable of a transition is associated with the log id. If the two variables are not associated, then the probability of firing the transition is equal in all logs. If the variables are associated, there exists at least one pair of logs with different transition probabilities.

The null hypothesis of the χ^2-test is that the variables are not associated. Thus, snKDiff conducts the test and rejects the null hypothesis when a p-value smaller than α is obtained.

If the null hypothesis is rejected, snKDiff performs a set of pairwise comparisons according to the $\mathcal{Z}$-test (Def. 6). Finally, snKDiff highlights a transition iff the null hypothesis of the $\mathcal{Z}$-tests is rejected for at least one pair of logs.

Example 2. *To demonstrate the above, assume that snKDiff compares a transition between k-PFSMs of three logs, i.e.,* t_i=(e_i,σ,e_i')*, for* $i \in \{1,2,3\}$*. Let us denote by* $V(e_i)$ *and* $V(t_i)$ *the number of visits to the source state and transition in the* i*th log resp. To conduct the* χ^2*-test, we construct two contingency tables, see Table I.*

TABLE I: Observed and expected values

Observed	log_1	log_2	log_3
t fired	$V(t_1)$	$V(t_2)$	$V(t_3)$
t skipped	$V(e_1) - V(t_1)$	$V(e_2) - V(t_2)$	$V(e_3) - V(t_3)$

Expected	log_1	log_2	log_3
t fired	$pV(q_1)$	$pV(q_2)$	$pV(t_3)$
t skipped	$(1 - p)V(q_1)$	$(1 - p)V(q_2)$	$(1 - p)V(q_3)$

The first table holds the observed values (O). The second holds the expected values (E) under the null hypothesis, which assumes that firing t *is unassociated with the log id. Thus, an equal firing probability is assumed in the models of all logs. Under this assumption, the probability of firing* t *equals* p=$\sum_{i\in\{1,2,3\}} V_i(t_i)/\sum_{i\in\{1,2,3\}} V_i(e_i)$.

In the tables, the first variable, which models if a transition was fired, takes two values (R={'success', 'failure'}), while the second variable, which models the log index, takes three values (C={1, 2, 3}). We compute the test statistics $x = \frac{\sum(O-E)}{E}$. *Then, we calculate the likelihood of obtaining the value of the test statistics or larger value* $P(\chi^2 > x)$ *for* χ^2*-distribution with* $(R-1)(C-1) = 2$ *degree of freedom. We refer to this value as the* p*-value of the test. If the* p*-value is smaller than* α*, we reject the null hypothesis and perform a set of* $\mathcal{Z}$*-tests between the logs, i.e., the following pairs are compared* (t_1, t_2), (t_2, t_3), (t_1, t_3) *for the same* α *value and user defined* d *value.*

As the number of differences found per transition by the pairwise comparisons may be large, it is useful to aggregate the results of multiple hypotheses tests by grouping the logs. Intuitively, a good grouping keeps logs with similar proportions together and separates logs with statistically different proportions. Such a grouping of log indices can be presented to the engineer to communicate the differences across logs. We present a simple and efficient label grouping procedure.

Log Index Grouping Procedure. First, we order the logs by their proportions. Then, we initialize a group (of logs) and add the logs to it, until reaching a log that is statistically different (for the chosen d, α) from at least one of the other logs in the group. When we reach such log, we initialize a new group and repeat the process, until all logs are processed.

Theorem 2. *The procedure ensures that no statistically different logs appear in the same group.*

Finally, we provide a high-level description of snKDiff.

The snKDiff Algorithm. The algorithm starts by constructing the k-DiffLFSM (see Def. 4). It enriches the states and transitions with the trace visit counters, one per log. Then, to find (k, d)-differences that are statistically significant, it iterates over its transitions. For each transition, it gets the number of source state and transition visits (n_i=$V(t(q))$, m_i=$V(t)$) per log, and conducts a χ^2-test. If the test result is a p-value lower than α, it continuous to perform a series of pairwise comparisons using the $\mathcal{Z}$-tests. If any of the $\mathcal{Z}$-test is found significant, it groups the log indices according to the log index grouping procedure. Then, it highlights the transition and labels it with the groups of log indices. Finally, snKDiff returns the labeled k-DiffLFSM, along with a summary of the results of the statistical tests preformed per transition (in a csv format).

Theorem 3 (snKDiff Soundness and Completeness)**.** *Let* k *be a positive integer,* $d \in [0, 1]$, $\alpha \in [0, 1]$*, and let* L_1, L_2, …, L_n *be* n *logs compared using snKDiff. Denote the logs' k-PFSMs by* S = {*k-PFSM*$_1$, *k-PFSM*$_2$, …, *k-PFSM*$_n$}*. Then, any reported* (k, d)*-difference corresponds to a statistically significant difference between a pair of models from* S *(soundness), and any* (k, d)*-difference between the models in* S *is reported (completeness).*

V. Evaluation

We present an evaluation in three parts. The first evaluates the effectiveness of s2KDiff and snKDiff in terms of soundness, completeness, and performance. The second is a controlled user study. The third is a case study using real-world logs.

All logs, models, and implementation code we describe together with their documentation are available for inspection and reproduction from [1].

A. Soundness and Performance Evaluation

We conducted an evaluation of the effectiveness of s2KDiff and snKDiff, guided by the following research questions:

RQA1 Do the expected statistical guarantees hold in practice? (statistical soundness)

RQA2 Are all differences reported? (statistical completeness)

RQA3 Do the algorithms scale?

Note that although we have theorems that state the soundness and completeness of s2KDiff and snKDiff, we still need to evaluate the correctness of the application of the statistical tests, i.e., to observe that the empirical error rate we obtain is below the theoretical α guarantee, and that the power of the test, β, increases with log size as expected. This may not be the case, e.g., if the assumption that the log traces are i.i.d. does not hold, or if there are bugs in the implementation of the algorithms.

Below we describe the corpus of models and logs we used, the different measures, the experiments, and their results.

1) Corpus of Models and Logs: In the evaluation we used 13 finite-state automaton models, taken from publicly available previously published works and reports: [11], [17], [21], [24], [29]–[31], [33]. The models varied in size and complexity: the alphabet size ranged from 8 to 22, the number of states ranged from 11 to 24, and the number of transitions ranged from 19 to 37. To facilitate the evaluation, we simplified the models by manually eliminating back-loops and self-loops. This simplification is not required for our algorithms to work. We use it to analytically calculate the seq_k transition probabilities for use as ground truth.

To produce a log from a model, we enriched the model with randomly selected transition probabilities and then created traces using a random trace generator. The trace generator starts on the dedicated initial state and makes a series of random transitions according to the transition probabilities, until reaching the dedicated terminal state.

2) Measures: We define two key measures for a set of hypothesis tests, the empirical statistical error rate $\hat{\alpha}$, and the empirical power of the test $\hat{\beta}$, for both the $\mathcal{Z}$-test and the χ^2-test, as follows. We use the classical true/false positive/negative measures, as summarized below.

$\mathcal{Z}$-test	$\mathcal{Z} \leq \alpha$	$\mathcal{Z} > \alpha$
$\lvert p_1 - p_2 \rvert < d$	fn	tn
$\lvert p_1 - p_2 \rvert \geq d$	tp	fp

χ^2-test	$\chi^2 \leq \alpha$	$\chi^2 > \alpha$
$\forall i, j\ p_i = p_j$	fn	tn
$\exists i, j\ p_i \neq p_j$	tp	fp

The error rate of a hypothesis test is defined w.r.t. the cases in which the null hypothesis holds. In the $\mathcal{Z}$-test, the null hypothesis holds when $|p_1 - p_2| < d$, and in the χ^2-test it holds when $\forall i, j\ p_i = p_j$. Thus, we define the empirical error rate of a series of statistical tests to be the proportion of rejected cases in which the null hypothesis held, i.e., $\hat{\alpha}_{\mathcal{Z}} = \hat{\alpha}_{\chi^2} = fp/(tn + fp)$. The empirical error rate $\hat{\alpha}$ is expected to be less than the statistical error rate of the test α.

The statistical power $\hat{\beta}$ of a hypothesis test is defined with regards to the cases in which the alternative hypothesis holds. Thus, we define the empirical power of a series of statistical tests to be the proportion of rejected cases in which the alternative hypothesis held, i.e., $\hat{\beta}_{\mathcal{Z}} = \hat{\beta}_{\chi^2} = tp/(tp + fn)$.

Lastly, we measure the absolute running times (in seconds). In measuring running times we included all steps, from parsing the logs, to computing the models, running the differencing procedures, and outputting the results. We executed all experiments on an ordinary laptop computer, Intel i5 CPU 2.4GHz, 8GB RAM with Windows 8 64-bit OS, Java 1.8.0_45 64-bit. We executed all runs 10 times, to average out measurement noise from the Java execution.

3) s2KDiff Experiments and Results: To answer the three research questions for s2KDiff we conducted the following experiment. For fixed k, d, and α, for each model, we produced two random logs of $n \in \{100, 1000, 2000\}$ traces, and measured $\hat{\alpha}_{\mathcal{Z}}$, $\hat{\beta}_{\mathcal{Z}}$, and the running time.

RQA1 & RQA2: Soundness and Completeness. We run s2KDiff with $k \in \{2, 3, 4\}$, $d \in \{0.01, 0.05, 0.1\}$, and $\alpha = 0.05$. We obtain $\hat{\alpha}_{\mathcal{Z}} < 0.05$ in all models, i.e., the the empirical error rate is below the theoretical α guarantee, as expected, see [1].

We obtained $\hat{\beta}_{\mathcal{Z}}$ medians of 0.38, 0.73, and 0.81, and averages of 0.39, 0.72, and 0.79, for n = 100, 1000, and 2000 resp. As can be seen, $\hat{\beta}_{\mathcal{Z}}$ increases with the log size. That is, the larger the log, the more differences the method is able to detect. This phenomena is well-known when conducting statistical tests.

RQA3: Performance. Figure 3 shows running times for k=2, d=0.01, α=0.05, $n \in \{2^6, 2^7, \ldots, 2^{14}\}$.

We report running times of k-Tails and s2KDiff. We observe that both algorithms scale well, and terminate within 1 second over the largest logs.

We computed the slowdown of s2KDiff w.r.t. k-Tails, by dividing the running time of s2KDiff by that of k-Tails. We obtained a median and an average slowdown of 10.90% and 11.75% resp. We did not identify any relation between the slowdown and the log size n.

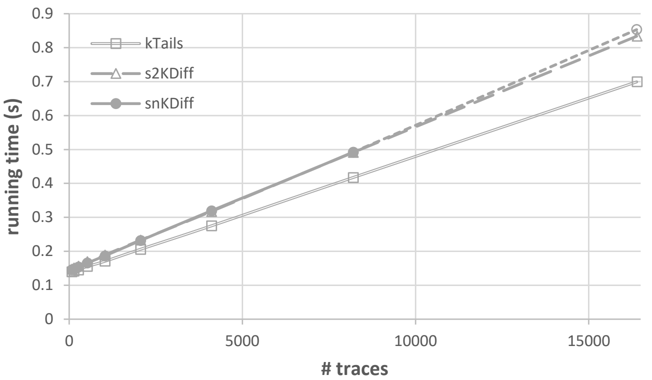


Fig. 3: The figure reports the average running times of kTails, s2KDiff, and snKDiff, when running with k=2, d=0.01, α=0.05, $n \in \{2^6 = 64, \ldots, 2^{14} = 16K\}$ over the 13 models. For snKDiff, $m = 2$.

Importantly, the running times increase linearly as the log size doubles with a median and an average increase of 15.30% and 24.34% for k-Tails, and 17.42% and 26.24% for s2KDiff resp. This occurs as the number of k-sequences quickly converges to a constant. As a result, only the traces reading time is affected by the increase of the log size.

We have evidence showing that s2KDiff achieves the statistical guarantees of its underlying statistical test α. Its power β varies across models, and increases dramatically with the log size. It incurs an acceptable overhead.

4) snKDiff Experiments and Results: To answer the three research questions for snKDiff we conducted the following experiment. For fixed k=2, d=0.1, and α=0.05, for each model, we produced m random logs for $m \in \{2, 4, 6, 8\}$, each log of length $n \in \{100, 1000, 2000\}$. Recall that snKDiff consists of a χ^2-test that is followed by a series of $\mathcal{Z}$-tests. The results of the $\mathcal{Z}$-tests were consistent with the results of s2KDiff, see [1]. We focus here on $\hat{\alpha}_{\chi^2}$, $\hat{\beta}_{\chi^2}$, and the running time.

RQA1 & RQA2: Soundness and Completeness. For fixed k=2, d=0.1, and α=0.05, for each model, we produced four ($m = 4$) random logs, each log of length $n \in \{100, 1000, 2000\}$, and run snKDiff over the different models.

We observe that $\hat{\alpha}_{\chi^2} = 0$ in all of the experiments. Recall that the null hypothesis in the χ^2-test is that the proportions between the transitions are equal. This was never the case, as we randomly assigned the transition probabilities. That is, in no experiments all four logs had equal probabilities over corresponding transitions.

We observe that $\hat{\beta}_{\chi^2}$ varies substantially across different models and increases with n. We did not observe any correlation w.r.t. m. We obtained $\hat{\beta}_{\chi^2}$ medians of 0.36, 0.88, and 0.96, and averages of 0.35, 0.83, and 0.93, for n = 100, 1000, and 2000 resp.

We repeated the same experiment for $k \in \{3, 4\}$ and $d \in \{0.05, 0.01\}$, and observed the same phenomenon.

RQA3: Performance. Figure 3 shows running times of snKDiff for k=2, d=0.01, α=0.05, $m = 2$, $n \in \{2^6, 2^7, \ldots, 2^{14}\}$.

Similar to s2KDiff, we observe that the algorithm scales, and terminates within 1 second over the largest logs. We measured median and average slowdown of snKDiff w.r.t. k-Tails of 9.34% and 11.03% resp. Similar to k-Tails and s2KDiff, the algorithm requires linear time w.r.t. the log size. The additional computation time required to apply the χ^2-test is negligible.

We run snKDiff for $k = 2$, $d = 0.01$, $\alpha = 0.05$, $n = 1024$, $m \in \{2, 4, 6, 8\}$, and observed linear increase in the running time across all models, see [1]. To test this further, we averaged the running time per $m \in \{2, 4, 6, 8\}$ across all models, and fit a linear line. We obtained $R^2 = 0.9979$. This indicates that a linear line captures the relation between the number of logs and the running time.

> We have evidence showing that snKDiff achieves the statistical guarantees of its underlying statistical tests $(\alpha_{\chi^2}, \alpha_{\mathcal{Z}})$. Its power w.r.t. both tests $(\beta_{\chi^2}, \beta_{\mathcal{Z}})$ varies across models, and increases dramatically with the log size. It incurs an acceptable overhead.

5) Threats to Validity: The selection of models in our evaluation may not represent typical systems. To mitigate, we used 13 publicly available models with non-trivial size and complexity, taken from previous works (see Sect. V-A1). Yet, we do not know to what extent these are representative of real-world systems. To generate logs from the models we used a publicly available trace generator [24]. It is possible that one may get different results if a different trace generator or a different coverage criterion is used.

B. Controlled User Study

We conducted a controlled user study to investigate whether s2KDiff and snKDiff are useful to their potential users. The study focuses on evaluating pertinent features of the algorithms and the research questions guiding it are:

RQB1 Can using s2KDiff and snKDiff help participants *more accurately* identify behavioral differences between different versions of the same system?

RQB2 Do s2KDiff and snKDiff *shorten the time required* for participants in identifying if and when a behavioral difference was introduced into a system?

1) Experiment Setup: **Independent and Dependent Variables.** The purpose of the user study is to examine whether s2KDiff and snKDiff provide participants with support in finding statistically significant differences among logs better than some alternatives (baselines), while considering a number of different logs. Thus, our experiment has two independent variables, the *tool* used to find log differences, the *log set*, and two dependent variables, *correctness* of the task solution (i.e., answers given by participants) and *completion time*.

Similar to our previous work [3], we compared s2KDiff + snKDiff, against k-Tails as the baseline. Since the classical k-Tails model does not include the information required to preform the statistical comparisons, we extend the model with transition probabilities, and with state, transition trace visit counters (see Sect. IV-A). We also allowed participants to use a text differencing tool [12]. We choose k-Tails, due to its popularity and for consistency with the previous work [3]. Direct comparison to 2KDiff + nKDiff [3] could not be done as these only account for existential differences.

We used four models: CVS, OrdSet, SMTPProtocol, and ZipOutStream (see Sect. V-A1). We chose these models as they include short and comprehensive alphabets and did not yield models that were too large for a human to interpret.

From each model, we generated five logs as follows. First we enriched the models with randomly generated transition probability distributions and copied them. Then, to introduce at least a single k-difference in the copied model, we randomly selected a pair of transitions with an identical source and mutated their probabilities by subtracting 0.2 from one and adding it to the other. We validated that the mutation resulted in valid transition probabilities. Then, we run the trace generator described in Sect.V-A over both models and produced three logs from the original model and two from the mutated one. All logs included 5000 traces.

Participants and Task Assignments. We invited 20 graduate students with background in computer science from two universities. All participants are senior students except one who is a first-year postgraduate. Every participant is required to perform four tasks by analyzing four log sets. For each task, a participant needs to use a log differencing tool to perform it. Among four tasks, two are required to use k-Tails, and the other two are required to use s2KDiff + snKDiff. All participants were presented with the log sets in a similar order. To avoid biases, we designed the experiment such that each log was analyzed by each of the tools.

Detailed Procedure. To complete a task, participants are required to analyze a log set using a specified tool and eventually answer several questions through a web interface. The following are the three questions that we asked participants for each task: *(1) Is there a 2-sequence whose frequency in one log is more than 0.1 higher than its frequency in another log and the difference is statistically significant? (2) Please specify such a 2-sequence*, and *(3) What is a pair of log ids whose corresponding logs exhibit such difference?*

The participants attempt each of the four tasks one by one. Before starting the user study, to instruct the participants in answering these questions and completing the tasks, they are required to read a tutorial and watch videos explaining the two log differencing tools and how they can be used to complete the tasks using an example task.

Note that if a participant answers ‘No’ to the first question, they will not be asked the subsequent questions. Our web interface recorded participants’ answers and the amount of time they used to complete each task.

2) Results: We report our user study results by answering the research questions mentioned earlier as follows.

RQB1: Correctness. After all participants completed the experiments, we evaluated the correctness of the participant

TABLE II: Percentage of participants who gave correct answers

	Q1		Q2		Q3	
	k-Tails	s2KDiff+snKDiff	k-Tails	s2KDiff+snKDiff	k-Tails	s2KDiff+snKDiff
CVS	60%	90%	10%	70%	10%	80%
ordSet	0%	100%	0%	90%	0%	90%
SMTPProtocol	30%	90%	10%	70%	20%	80%
ZipOutStream	40%	90%	10%	80%	10%	90%

TABLE III: Average completion time results (seconds)

	k-Tails	s2KDiff+snKDiff		k-Tails	s2KDiff+snKDiff
CVS	843.50	380.00	**SMTPProtocol**	600.90	317.10
ordSet	522.70	458.70	**ZipOutStream**	562.10	179.90

answers. If a participant chose "No" option for the first question of a task, the remaining two questions are labeled as incorrect. Table II presents the percentage of participants who gave correct answers. For question 1, it is the percentage of participants who answered "Yes". As shown in the table, very few participants who use k-Tails can identify the difference among the log sets. Even though some of them answered "Yes" for the first question, they wrongly identified the 2-sequence and the log pairs. This is because the models generated by k-Tails are too complex, with many transitions. It is difficult for the participants to switch different browser windows to compare several models. On the other hand, most of the participants who use s2KDiff + snKDiff gave the correct answers for question 2 and 3. Only few participants did not answer the questions correctly; we talked with them and found that they did not fully understand the tool as it is the first time they used it.

RQB2: Completion Time. Table III presents the average completion time for each task using k-Tails and s2KDiff + snKDiff. The average completion time for tasks performed using s2KDiff + snKDiff is lower than that of k-Tails. We also performed Wilcoxon signed-rank test and found that the differences are all statistically significant at a confidence level of 99% with a large effect size.

C. Case Study: findyourhouse.com from Ghezzi et al. [15]

We conducted a case study in order to answer the following research question:

RQC What kind of insights about the evolution of a system over time and about differences of its usage patterns can one learn by using our tools?

We used a log from the real-world application, *findyourhouse.com*, presented by Ghezzi et al. [15]. To parse the log, we followed the description of the authors and the regular expressions included in the paper. In this log, each trace includes the pages visited in a single user session. The log represents data collected over a year of operation with real users. The alphabet size (unique web-pages) is 26. The total number of events is 37,465. The results and figures of the usage scenarios below are provided in [1].

1) Changes in behavior over time: We used the session date to split the original log into 4 disjoint logs, each consisting of sessions from one quarter (3 consecutive months). The 4 logs contained 2048, 2606, 2415, 1657 traces resp. The average trace length is 4.29.

We then run snKDiff on the 4 logs, with k=1, d=0.05, and α=0.05. Executing snKDiff took 2.01 seconds. Out of 335 transitions, snKDiff reported 49 to have statistically significant difference between at least 2 logs.

We focused on the sales search module, and specifically on its first four result pages. We selected this module because it is the main module of the application, it was heavily used by its users, and it was analyzed by Ghezzi et al. [15]. From the snKDiff output, when considering the statistically significant transitions, we observed the following phenomenon: as time progresses from one quarter to the next, users are more likely to end their session on earlier search result pages. This may indicate a possible improvement in the quality of the sales search results, a change in the order of the search results, or any other change (we were unable to contact the original owners of the data for comments on these findings).

As evidence, snKDiff found a statistically significant difference in the transition probabilities from the first search page to the second, where p_{q_1}=35.1%, p_{q_2}=32.8%, p_{q_3}=26.5%, p_{q_4}=15.53% in quarters 1 to 4 resp. Further, the transition probabilities of terminating the session after the first search page were 7.5%, 9.09%, 16.0%, and 30.0% in quarters 1 to 4 resp., and were also statistically different.

Note that since all transitions have been observed at least once in each of the 4 logs, the algorithms of [3] could not reveal this phenomenon. Also note that the analysis by Ghezzi et al. [15], which presented this log, deals with transition frequencies but not with their differences and not with statistical significance. Thus, it too, could not reveal this phenomenon.

2) Desktop vs. mobile users - usage preference: We used the user's browser and operating system (as documented in the log) to separate traces of desktop and mobile users, and thus produced two logs of 5033 and 885 traces resp. (ignoring traces from bots). The events alphabet size for the desktop and the mobile logs were 16 and 13 resp. Average trace length was 6.22 and 3.81 resp. For s2KDiff below we used $k = 1$, $d = 0.05$, and $\alpha = 0.05$. Executing s2KDiff took 1.05 seconds.

Out of 145 transitions, s2KDiff reported 14 (k, d)-differences. Interestingly, the mobile users show higher probability of transitioning out of the application in comparison to the desktop users in 6 out of the 13 common web-pages: `homepage`, `sales_anncs`, `renting_anncs`, `renting_page`, `contacts`, `news_article`. For example, exiting through the `homepage` had a transition probability of 23.68% and 39.9% in desktop and mobile users resp. The trace selected for the mobile users included a single visit to the `homepage`. Indeed, out of all mobile traces, this trace appeared 108 times. These results are also consistent with the lower number of webpage visits per trace in the mobile log.

This demonstrates how s2KDiff reveals significant differences between the behaviors of desktop and mobile website users, which could not be revealed using previous methods.

VI. Discussions

Implications and Broader Context. s2KDiff and snKDiff are not meant to replace prior log differencing tools, e.g., our own 2KDiff and nKDiff [3]. Rather, they complement the prior work for use cases where differences in frequencies are meaningful.

A use case where the capabilities of the new tools are needed is highlighted in Section V-C. In that case study, s2KDiff and snKDiff have been employed to uncover insights about the evolution of a system over time, and about differences of its usage patterns. 2KDiff and nKDiff are unable to uncover such insights as they do not analyze differences in frequencies. Another potential application of log differencing is in the area of malware analysis; log differencing can be applied to compare logs of clean and infected apps to identify malicious activities. The power of s2KDiff and snKDiff to consider differences in frequencies will be beneficial to uncover hidden malicious behaviors that are masqueraded as valid ones, e.g., [36].

Dealing with Unstructured Logs. Our work, like most related work on model inference from logs, requires structured logs as input. In practice, many logs are unstructured. Fortunately, past studies have proposed preprocessing methods to convert unstructured to structured logs, e.g., [13], [26], [39]. Those methods may be employed to allow our approach to work for unstructured logs.

Configuring s2KDiff and snKDiff. Our tools take as input a structured log and 3 parameters: α, k, and d. The choice of α, the significance level, is clear, as it is considered a standard; many past studies use 0.01 or 0.05 as significance levels for their statistical tests. The choice of k, the parameter for the k-Tails algorithm, is also clear; past studies often set the value of k to be 1 or 2 [9], [23], [25]; indeed, any use of k-Tails requires the engineer to choose a value for k. The actual parameter whose setting is to be determined is d, which ranges between 0 and 1. To set this parameter, engineers can start with a high value of d, close to 1, and gradually decrease it until differences are uncovered. If more differences are desired, the value of d can be decreased further. The same strategy is often employed by users of rule-based or scenario-based specification mining tools, e.g., [8], [20], [22], who need to configure several parameters, e.g., support and confidence.

VII. Related Work

Many works suggest means to infer models from logs. The works differ in the kinds of input logs and output models. A number of them mined specifications in the form of FSM from a set of logs. For example, Beschastnikh et al. proposed Synoptic, which infers three kinds of temporal invariants from logs and uses them to refine an inferred FSM [5]. Le et al. proposed SpecForge, which forges a new FSM from FSMs mined by other solutions through model fission and fusion operations [18]. More recently, Le and Lo proposed DSM that uses the power of a deep learning engine (i.e., Long Short-Term Memory network) to learn FSMs from logs [19]. Different from the aforementioned works, in this work, we focus on producing an FSM capturing differences between two logs, s2KDiff, and between many logs at once, snKDiff.

In an experience report on log-based behavioral differencing, Goldstein et al. [16] focused on visualizing anomalies. Given two logs, they use k-Tails to build a model for each log, and then compare the two models. Their models are enriched with quantitative data but their analysis does not involve statistical guarantees. Their work is limited to comparing two logs while our snKDiff algorithm compares many logs at once.

Wang et al. [37] compared sets of inferred temporal invariants, as one of several approaches to examine whether tests are representative of field behavior. The work does not consider the statistical significance of the differences it finds. Comparing lab and field logs is one potential application of our work.

Most related is our recent work [3], which presented 2KDiff and nKDiff. As we discussed earlier, here we extend these algorithms fundamentally. Specifically, unlike [3], our present work accounts for the frequencies of behaviors in the logs and provides statistical guarantees about the differences it finds.

In Sect. IV-C we presented a procedure for grouping log labels. Scott and Knott [32] proposed the Scott-Knott algorithm, a clustering based approach, which compares multiple variables and breaks them into homogeneous groups, i.e., groups with statistically distinct means. Similarly, Tantithamthavorn et al. [34] proposed the Scott-Knott Effect Size Difference Test, which also accounts for the difference between the means of the groups by incorporating the effect-size measure. In contrast, our underlying hypothesis test is different, as it also accounts for the distance between the proportions. Further, to avoid clutter and emphasize the differences, we only include labels of logs that appear statistically different from at least one of the other logs.

VIII. Conclusion and Future Work

We presented statistical log differencing, as a means to compute and present differences between two or more execution logs, taking into account the frequencies of the behaviors in the logs and reporting only (and all) statistically significant differences. Our evaluation shows the effectiveness of our work in terms of soundness, completeness, and performance. It demonstrates effectiveness using a controlled user study and potential applications via a case study using real-world logs.

We consider the following future work. First, for additional statistical confidence and scalability over logs of unbounded length, one may consider adapting the approach of our previous work on model inference with statistical guarantees [7] to add another statistical dimension, i.e., allowing engineers to set a required confidence level and then sample from the logs at hand until reaching statistical guarantees about the confidence in the result (where the probability of additional sampling to reveal new significant differences is smaller than a required threshold). Second, additional case studies to evaluate the use of statistical log differencing in practice and the kinds of insights one can learn from its application. We specifically consider applications in test generation and execution.

Acknowledgments

We thank the reviewers for their helpful comments. This work has been partly supported by the Blavatnik Interdisciplinary Cyber Research Center at Tel Aviv University, by the Singapore National Research Foundation's National Cybersecurity Research & Development Programme (No. NRF2016NCR-NCR001-008), and by NSFC Program (No. 61902344).

References

[1] Supporting materials website. http://smlab.cs.tau.ac.il/xlog/#ASE19b.

[2] A. Agresti. *An introduction to categorical data analysis*. Wiley, New York, 1996.

[3] H. Amar, L. Bao, N. Busany, D. Lo, and S. Maoz. Using finite-state models for log differencing. In *ESEC/SIGSOFT FSE*, pages 49–59. ACM, 2018.

[4] I. Beschastnikh, Y. Brun, J. Abrahamson, M. D. Ernst, and A. Krishnamurthy. Using declarative specification to improve the understanding, extensibility, and comparison of model-inference algorithms. *IEEE Transation on Software Engineering*, 41(4):408–428, 2015.

[5] I. Beschastnikh, Y. Brun, S. Schneider, M. Sloan, and M. D. Ernst. Leveraging existing instrumentation to automatically infer invariant-constrained models. In *Proceedings of the 19th ACM SIGSOFT Symposium on the Foundations of Software Engineering and the 13th European Software Engineering Conf. (ESEC/FSE)*, pages 267–277, 2011.

[6] A. W. Biermann and J. A. Feldman. On the synthesis of finite-state machines from samples of their behavior. *IEEE Transactions on Computers*, 21(6):592–597, June 1972.

[7] N. Busany and S. Maoz. Behavioral log analysis with statistical guarantees. In *Proceedings of the 38th ACM/IEEE Int. Conf. on Software Engineering (ICSE)*, pages 877–887. ACM, 2016.

[8] Z. Cao, Y. Tian, T. B. Le, and D. Lo. Rule-based specification mining leveraging learning to rank. *Autom. Softw. Eng.*, 25(3):501–530, 2018.

[9] H. Cohen and S. Maoz. Have we seen enough traces? In M. B. Cohen, L. Grunske, and M. Whalen, editors, *ASE*, pages 93–103. IEEE, 2015.

[10] J. E. Cook and A. L. Wolf. Discovering models of software processes from event-based data. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 7(3):215–249, 1998.

[11] V. Dallmeier, N. Knopp, C. Mallon, G. Fraser, S. Hack, and A. Zeller. Automatically generating test cases for specification mining. *IEEE Trans. Software Eng.*, 38(2):243–257, 2012.

[12] Diffchecker. http://www.diffchecker.com.

[13] M. Du, F. Li, G. Zheng, and V. Srikumar. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proc. of the 2017 ACM SIGSAC Conf. on Computer and Communications Security, CCS*, pages 1285–1298, 2017.

[14] M. Ernst, J. Cockrell, W. Griswold, and D. Notkin. Dynamically discovering likely program invariants to support program evolution. *TSE*, 27(2):99–123, 2001.

[15] C. Ghezzi, M. Pezzè, M. Sama, and G. Tamburrelli. Mining behavior models from user-intensive web applications. In *Proceedings of the Int. Conf. on Software Engineering*, ICSE '14, pages 277–287. ACM, 2014.

[16] M. Goldstein, D. Raz, and I. Segall. Experience report: Log-based behavioral differencing. In *Proceedings of the 28th IEEE Int. Symposium on Software Reliability Engineering (ISSRE)*, pages 282–293. IEEE Computer Society, 2017.

[17] W. Grieskamp, N. Tillmann, and M. Veanes. Instrumenting scenarios in a model-driven development environment. *Information & Software Technology*, 46(15):1027–1036, 2004.

[18] T. B. Le, X. D. Le, D. Lo, and I. Beschastnikh. Synergizing specification miners through model fissions and fusions (T). In *30th IEEE/ACM Int. Conf. on Automated Software Engineering, ASE*, pages 115–125, 2015.

[19] T. B. Le and D. Lo. Deep specification mining. In *Proceedings of the 27th ACM SIGSOFT Int. Symposium on Software Testing and Analysis, ISSTA*, pages 106–117, 2018.

[20] C. Lemieux, D. Park, and I. Beschastnikh. General LTL specification mining (T). In *30th IEEE/ACM Int. Conf. on Automated Software Engineering, ASE*, pages 81–92, 2015.

[21] D. Lo and S.-C. Khoo. Quark: Empirical assessment of automaton-based specification miners. In *WCRE*, pages 51–60. IEEE Computer Society, 2006.

[22] D. Lo and S. Maoz. Scenario-based and value-based specification mining: better together. *Automated Software Engineering*, 19(4):423–458, 2012.

[23] D. Lo, L. Mariani, and M. Pezzè. Automatic steering of behavioral model inference. In *Proceedings of the 7th joint meeting of the European Software Engineering Conf. and the ACM SIGSOFT Int. Symposium on Foundations of Software Engineering*, pages 345–354, 2009.

[24] D. Lo, L. Mariani, and M. Santoro. Learning extended FSA from software: An empirical assessment. *Journal of Systems and Software*, 85(9):2063–2076, 2012.

[25] D. Lorenzoli, L. Mariani, and M. Pezzè. Automatic generation of software behavioral models. In *Proceedings of the 30th ACM/IEEE Int. Conf. on Software Engineering (ICSE)*, pages 501–510, 2008.

[26] J. Lou, Q. Fu, S. Yang, Y. Xu, and J. Li. Mining invariants from console logs for system problem detection. In *USENIX*, 2010.

[27] L. Mariani, F. Pastore, and M. Pezzè. Dynamic analysis for diagnosing integration faults. *IEEE Transactions on Software Engineering*, 37(4):486–508, 2011.

[28] M. J. S. Morris H. DeGroot. *Introduction to Probability and Statistics*. Pearson; 4 edition (January 6, 2011), 2011.

[29] S. Mouchawrab, L. C. Briand, Y. Labiche, and M. Di Penta. Assessing, comparing, and combining state machine-based testing and structural testing: A series of experiments. *IEEE Trans. Softw. Eng.*, 37(2):161–187, Mar. 2011.

[30] E. Poll and A. Schubert. Verifying an implementation of SSH. In *WITS*, volume 7, pages 164–177, 2007.

[31] J. Postel. Transmission control protocol. RFC 793, Internet Engineering Task Force, September 1981.

[32] A. Scott and M. Knott. A cluster analysis method for grouping means in the analysis of variance. *Biometrics*, 30, 09 1974.

[33] J. Sun and J. Song Dong. Design synthesis from interaction and state-based specifications. *IEEE Trans. Softw. Eng.*, 32(6):349–364, June 2006.

[34] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto. An empirical comparison of model validation techniques for defect prediction models. *IEEE Transactions on Software Engineering*, 43(1):1–18, Jan 2017.

[35] E. Vidal, F. Thollard, C. de la Higuera, F. Casacuberta, and R. C. Carrasco. Probabilistic finite-state machines-part i. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(7):1013–1025, July 2005.

[36] D. A. Wagner and P. Soto. Mimicry attacks on host-based intrusion detection systems. In *Proceedings of the 9th ACM Conf. on Computer and Communications Security, CCS 2002, Washington, DC, USA, November 18-22, 2002*, pages 255–264, 2002.

[37] Q. Wang, Y. Brun, and A. Orso. Behavioral execution comparison: Are tests representative of field behavior? In *Proceedings of the IEEE Int. Conf. on Software Testing, Verification and Validation (ICST)*, pages 321–332. IEEE Computer Society, 2017.

[38] N. Weiss. *Introductory Statistics*. Pearson Education, 2012.

[39] X. Yu, P. Joshi, J. Xu, G. Jin, H. Zhang, and G. Jiang. Cloudseer: Workflow monitoring of cloud infrastructures via interleaved logs. In *Proc. of the 21st Int. Conf. on Architectural Support for Programming Languages and Operating Systems, ASPLOS*, pages 489–502, 2016.