Have We Seen Enough Traces?

Hila Cohen and Shahar Maoz
School of Computer Science
Tel Aviv University, Israel

***Abstract*—Dynamic specification mining extracts candidate specifications from logs of execution traces. Existing algorithms differ in the kinds of traces they take as input and in the kinds of candidate specification they present as output. One challenge common to all approaches relates to the faithfulness of the mining results: how can we be confident that the extracted specifications faithfully characterize the program we investigate? Since producing and analyzing traces is costly, how would we know we have seen enough traces? And, how would we know we have not wasted resources and seen too many of them?**

In this paper we address these important questions by presenting a novel, black box, probabilistic framework based on a notion of log completeness, and by applying it to three different well-known specification mining algorithms from the literature: k-Tails, Synoptic, and mining of scenario-based triggers and effects. Extensive evaluation over 24 models taken from 9 different sources shows the soundness, generalizability, and usefulness of the framework and its contribution to the state-of-the-art in dynamic specification mining.

I. Introduction

Much literature has been published on dynamic specification mining, which extracts candidate specifications from logs of program execution traces. Different approaches suggest different algorithms, which differ in the kinds of traces they take as input and in the kinds of candidate specification they present as output (in terms of content, expressive power, and format). Example approaches include [5], [13]–[15], [17], [18], [21], [22], [26], [27], [29]–[31], [37], [38], [41], [42].

One challenge common to all approaches relates to the faithfulness of the mining results: how can we be confident that the extracted specifications faithfully characterize the program we investigate? As the mined specifications may be used for comprehension, test generation, and verification, their faithfulness to the program we investigate is important. Yet, producing and analyzing traces is costly, so how would we know we have seen enough of them? And, how would we know we have not wasted resources and seen too many of them?

In this paper we address these important questions by presenting a novel, black box, probabilistic framework based on a notion of log completeness. We consider logs of execution traces extracted from a running system under investigation. Intuitively, a log is complete with regard to a specific system and a mining algorithm, if adding any new system trace to the log will not change the output of the algorithm. In dynamic specification mining, the system and its full set of system traces is unknown. Therefore, given a log of system traces, we estimate the probability that the log is complete. We say that this estimation is the log's *confidence*.

A log's confidence may be practically used as follows. If not many traces are available, one can compute the confidence of the available log in order to estimate the expected faithfulness of the mining results. A low confidence of, say, 0.2, hints that the mining results may be far off from characterizing the behavior of the system under investigation. A very high confidence of, say, 0.95, hints that the mining results are probably very close to correctly characterize the behavior of the system under investigation. If producing and analyzing more traces is possible but costly, one can set a high confidence threshold of, say, 0.85, and stop adding new traces when the threshold is reached.

We apply the new framework to three different well-known dynamic specification mining algorithms from the literature: k-Tails [6], Synoptic [5], and mining of scenario-based triggers and effects [25]. Inspired by InvariMint [4], we represent each algorithm using a set of log properties. For k-Tails, we use a property describing the existence of possible sequences of length k in the log. For Synoptic, we use four different properties, corresponding to the k-Tails property with $k = 2$ and to the three temporal invariant properties used by the Synoptic algorithm. For triggers and effects, we use a single three-valued property representing the possible relation between the trigger and all of the possible effects (or vice versa). The framework assumes that mining randomly and independently samples traces from the log. For all three algorithms, we compute the confidence of a set of traces, by estimating the probability that it manifests the complete properties that any log of the system under investigation can manifest. Each log property requires a different probabilistic estimation.

An interesting feature of our notion of completeness is that it is relative to a system under investigation and a specific mining algorithm: a log may be complete with regard to one algorithm and incomplete with regard to another. Yet, computing a log's confidence with regard to a specific algorithm does not require running the specification mining algorithm itself.

An important feature of our notion of confidence is its non-monotonicity: adding traces to a log does not guarantee an increase in confidence and may even decrease it (still, a log's confidence doesn't depend on the order of traces in it). We discuss this apparently counter-intuitive feature in Sect. VII.

Finally, we present an extensive evaluation of our work, including the results of experiments with 24 models of real-world systems taken from publicly available previously published works. The results show that for all three algorithms, the probability that a log is complete can be estimated efficiently and be effectively applied, with high reliability, to improve the use of the dynamic specification mining algorithm and the confidence one may have in its results.

In recent preliminary work [8] (ASE'14 New Ideas Track) we have presented a notion of confidence computation for k-Tails, termed k-confidence, with preliminary evaluation of its effectiveness. Our present paper extends this previous work significantly, and makes the following contributions:

- The presentation of a general framework for log confidence and its application, beyond k-Tails, to two additional well-known specification mining algorithms;
- An extensive evaluation over 24 real-world models, which provides strong evidence for the correctness and effectiveness of the confidence computations;
- An implementation available for reproduction and extension for additional specification mining algorithms.

Sect. II presents examples for the use of log confidence, for Synoptic and for mining triggers and effects. Sect. III presents the formal foundations of the log completeness framework and Sect. IV presents its application to the three dynamic specification mining algorithms. Sect. V presents an example confidence computation. Sect. VI presents the evaluation, Sect. VII discusses design decisions and limitations, Sect. VIII discusses related work, and Sect. IX concludes.

II. Examples

We use small examples to demonstrate the usage of log confidence. All traces and models discussed in the examples below are available in [1]. The presentation of the examples is semi-formal, for illustrative purposes.

A. Example I

In [5], the authors use an example of a shopping cart to demonstrate how Synoptic is able to reveal a bug where the user can use an invalid coupon to reduce the price. To reveal this bug, the problematic behavior must appear in the traces.

Consider an engineer having a log of 34 traces of the shopping cart system, adapted from the example traces of [5]. Fig. 1 shows the model suggested by Synoptic for this log (the dashed transition `invalid-coupon` to `reduce-price` is *not* part of the model suggested by Synoptic for this 34 traces log). The model does not include the bug so at this stage, the user may wrongly conclude that the shopping cart system has no bugs. How would she know if more traces are needed?

Indeed, adding 15 more traces, and feeding the resulting log of 49 traces to Synoptic, results in a revised model as shown in Fig. 1 (including the dashed transition), which reveals the invalid coupon bug.

Our tool computes a confidence of 0.59 to the first log and of 0.98 to the second log. This confidence is essentially an estimation for the probability that the log is complete, computed solely based on the traces themselves. If the user has set a minimum confidence threshold of 0.95 (which is a very safe threshold according to our evaluation), she would not have stopped analyzing traces too soon, i.e., before finding the bug.

Should she continue to analyze more traces? Given the computed confidence of 0.98, the probability that additional traces will reveal new behaviors is very small. Indeed in our example, an extension of the log with 22 additional traces (all new traces, not duplicates of any of the traces seen before) resulted in a slightly higher confidence of 0.99 but in the *same* model as was suggested by Synoptic for the 49 traces. Thus, by stopping the analysis when confidence reached 0.95 the engineer saved the resources required in order to produce and analyze the additional traces, yet did not lose any information.

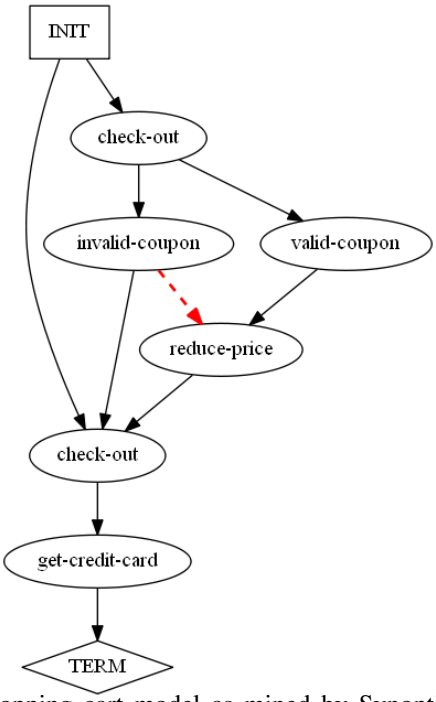


Fig. 1: Shopping cart model as mined by Synoptic from the log of 34 traces (without the dashed transition) and from the log of 49 traces (including the dashed transition)

B. Example II

In mining scenario-based triggers and effects [25], the engineer provides a trigger scenario (a sequence of events) as input, and receives a set of candidate effects (sequences of events) as output, with the following semantics: whenever the trigger scenario occurs in the traces (in the specified events order but possibly with other events interleaved), eventually each of the candidate effects occurs too (again, in the specified events order but possibly with other events interleaved).

In [17], the authors use an example set of execution traces from *CrossFTP*, an open-source FTP server [11]. The full data set, available from [16], contains 54 traces with an average trace length of 34, over an alphabet of 50 events.

Consider an engineer investigating the behavior of the CrossFTP server, looking for scenario-based effects to the trigger consisting of a call to open a new connection and a call to transfer data (used in CrossFTP both in the context of upload and in the context of download):
`ConnectionManagerImpl.newConnection(...)`
`RequestHandler.transfer(...)`.

When the engineer executed the mining with this trigger, on a sub log of randomly selected 24 traces from the full log, more than 12 unique candidate effects were reported. For

example, one of these candidate effects was the scenario: `ConnectionManagerImpl.closeConnection(...)` `ConnectionManagerImpl.dispose(...)`.

However, none of these candidate effects truly characterizes the behavior of the CrossFTP. When executing mining of triggers and effects with the same trigger on the full log of 54 traces, no effect is reported, since a few traces include multiple calls to the `transfer(...)` method with no occurrence of an effect between them. Indeed, in CrossFTP, no scenario exists which always occurs in this log after the given trigger.

Our tool computes a confidence of 0.2 to the 24 traces log, and a confidence of 0.98 to the 54 traces log. Again, this confidence is essentially an estimation for the probability that the log is complete, computed solely based on the traces themselves. Indeed, mining scenario-based triggers and effects from the log with the low confidence produced results that do not characterize the true behavior of the system under investigation. In other words, the results of mining from a log that has a low confidence should not be trusted.

C. Running Example

We take the `java.util.zip.ZipOutputStream` model from [20] as a small running example for this paper. The model consists of 3 states over an alphabet of 5 events. We consider a randomly generated log for this model, consisting of 7 traces. See Fig. 2.

III. Log Completeness Framework

A. Basic Definitions

A trace over an alphabet Σ is a finite word $\sigma = \langle e_1, e_2, \ldots, e_m \rangle$ where $e_1, \ldots, e_m \in \Sigma$. For $j \geq 1$ we use $\sigma(j)$ to denote the jth element in σ.

Let M be a model over an alphabet Σ. We use $T(M) \subseteq \Sigma^*$ to denote all traces accepted by the model M. A log of M, $l \subseteq T(M)$ is a finite set of traces from $T(M)$. We denote the set of all possible logs of M by $L(M)$.

The basic definition underlying log completeness is the log-property LP, consisting of a pair of functions. The first function, LP_{tr}, maps every trace σ and a sequence of events es to a value in some domain D_{tr}. Intuitively, LP_{tr} assigns a value to the relation between the property LP and the trace. The second function, LP_{log}, maps subsets of D_{tr} to D_{log}; it aggregates the trace level results to the log level.

Example 1. If $D_{tr} = D_{log} = \{0, 1\}$, LP_{tr} may be used to represent whether a property holds in the trace, and LP_{log} may represent whether it holds in an entire log. An example property is an invariant of the form 'a always precedes b'.

Definition 1 (log-property)**.** A log-property $LP = \langle LP_{tr}, LP_{log} \rangle$ is a pair of functions $LP_{tr} : \Sigma^* \times \Sigma^i \to D_{tr}$ and $LP_{log} : \mathcal{P}(D_{tr}) \to D_{log}$.

We say that a log $l \in L(M)$ is complete with regard to a log-property LP if the information one may extract about the property from the log l is equal to the information one may extract about it from any log that includes l (and thus specifically from all the traces in $T(M)$). Formally:

Definition 2 (LP-log-completeness)**.** A log $l \in L(M)$ is complete with regard to a log-property $LP = \langle LP_{tr}, LP_{log} \rangle$ iff $\forall l' \in L(M)$ s.t. $l \subseteq l'$, $\forall es \in \Sigma^i$ $LP_{log}(\{LP_{tr}(\sigma, es) | \sigma \in l\}) = LP_{log}(\{LP_{tr}(\sigma, es) | \sigma \in l'\})$.

We now lift the completeness definition from properties to algorithms. A specification mining algorithm A accepts a log $l \in L(M)$ as input and outputs a candidate model $A(l) = M'$.

We represent an algorithm A by a set of log properties $LP(A) = \{LP^1, LP^2, \ldots, LP^k\}$. We say that a log l is complete with regard to an algorithm A iff l is complete with regard to each of the log-properties of A. Formally:

Definition 3 (A-log-completeness)**.** A log $l \in L(M)$ is complete with regard to an algorithm A iff $\forall LP \in LP(A)$, l is complete with regard to LP.

Note that to make the framework useful, one should define the set of log-properties $LP(A)$ such that, if $l \in L(M)$ is complete with regard to A, adding any trace $\sigma \in T(M)$ to l will not affect the candidate model A computes for l (indeed this is part of our evaluation, see Sect. VI-B). Also note that log completeness is defined relative to an algorithm (specifically a set of log-properties). The same log may be complete with regard to one algorithm and incomplete with regard to another algorithm.

B. Estimating Log Completeness

Given an algorithm A and a log $l \in L(M)$, our goal is to estimate the probability that l is complete with regard to A, i.e., to compute l's confidence. To do this, we compute l's confidence with regard to the LPs in $LP(A)$ and take the minimum (intuitively, choosing the minimum is a conservative choice).

For each $LP \in LP(A)$, we define a random variable $Y_{LP}(\sigma)$ over $\Omega = T(M)$, which maps a trace to its property results. Formally:

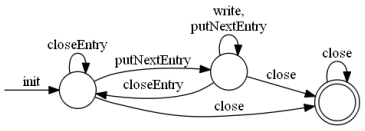
$$Y_{LP}(\sigma) : Y_{LP}(\sigma)[es] = LP_{tr}(\sigma, es).$$

Example 2. We denote the invariant property 'a always precedes b' mentioned above by $LP^{\leftarrow}$. For the trace tr_2 of our running example in Fig. 2 and the sequences $\langle init, closeEntry \rangle$ and $\langle putNextEntry, closeEntry \rangle$, we have $Y_{LP^{\leftarrow}}(tr_2)[\langle init, closeEntry \rangle] = 1$ and $Y_{LP^{\leftarrow}}(tr_2)[\langle putNextEntry, closeEntry \rangle] = 0$, because the first holds in the trace while the second does not.

For $y \in D_{tr}^{|\Sigma|^i}$ we denote the probability that Y_{LP} equals y by $\pi_{LP}(y)$:

$$\pi_{LP}(y) = \mathbb{P}[Y_{LP} = y].$$

Example 3. To continue our example above, where $D_{tr} = D_{log} = \{0, 1\}$, for the invariant property 'a always precedes b', we have $i = 2$ and so y can be viewed as a $\Sigma \times \Sigma$ 2-dimensional array (a matrix) over $\{0, 1\}$. In our running example the alphabet size is $|\Sigma| = 5$ and so for this property we will use a 5×5 matrix (see later in Table I).



tr_1: `init,closeEntry,close,close`
tr_2: `init,closeEntry,putNextEntry,closeEntry,putNextEntry,closeEntry,close,close`
tr_3: `init,putNextEntry,write,putNextEntry,putNextEntry,closeEntry,close`
tr_4: `init,putNextEntry,write,close`
tr_5: `init,close`
tr_6: `init,putNextEntry,write,close,close`
tr_7: `init,closeEntry,putNextEntry,write,closeEntry,closeEntry,putNextEntry,close`

Fig. 2: The running example model of `java.util.zip.ZipOutputStream` from [20], and 7 randomly generated traces used to demonstrate confidence computation.

$\pi_{LP}(y)$ is the probability that in a random trace from $T(M)$ we get the values of LP as they are encoded in y. It is determined by M but M is considered unknown.

We consider all traces from a log l to be samples from Y_{LP}. We assume that traces are randomly and independently chosen from $T(M)$. If $|l| = n$ we denote them by $Y_{LP1}, Y_{LP2}, \ldots, Y_{LPn}$. These are independent, identically distributed random variables, versions of Y_{LP}. Another random variable we define is Y^n_{LP}, which aggregates all these samples to the property values for the entire log:

$$Y^n_{LP} : Y^n_{LP}[es] = LP_{log}(\{Y_{LPi}[es])|1 \leq i \leq n\})$$

We now define the true, but unknown, log-property values, $f_{LP}(\pi)$, in order to later compute the probability that Y^n_{LP} is equal to it:

$$f_{LP}(\pi) : f_{LP}(\pi)[es] = LP_{log}(\{y[es]|\pi(y) > 0\}).$$

Example 4. For the running example model in Fig. 2, the log property 'always precedes' $LP^{\leftarrow}$, and the sequence $\langle putNextEntry, closeEntry\rangle$, since there are traces in $T(M)$ where the invariant is not violated (e.g., tr_3) and others where it is violated (e.g., tr_2), the property does not hold in the log and so $f_{LP^{\leftarrow}}(\pi)[\langle putNextEntry, closeEntry\rangle] = 0$.

Note that M determines $f_{LP}(\pi)$. We can now write Def. 3 using the above notation as follows: a log l is complete with regard to an algorithm A iff

$$\forall LP \in LP(A).Y^n_{LP} = f_{LP}(\pi).$$

Recall that our goal is to estimate the probability that l is complete with regard to A. Using the notation defined above, what we are looking to estimate is $\mathbb{P}[Y^n_{LP} = f_{LP}(\pi)]$.

The above represents l's confidence with regard to a single log-property LP. We define the confidence of a log l with regard to an algorithm A, to be the minimum of l's confidence with regard to all LPs in $LP(A)$. Formally:

Definition 4 (A-log-confidence)**.** The confidence of a log $l \in L(M)$ with regard to an algorithm A is
$min\{\mathbb{P}[Y^n_{LP} = f_{LP}(\pi)]|LP \in LP(A)\}$.

IV. Log Completeness Applied

We now present the application of the framework to three different previously published dynamic specification mining algorithms: k-Tails [6], Synoptic [5], and mining of scenario-based triggers and effects [25]. For each algorithm we define the relevant log-properties $LP(A)$ and show the estimation of $\mathbb{P}[Y^n_{LP} = f_{LP}(\pi)]$ for each $LP \in LP(A)$.

A. k-Tails

k-Tails [6] is a dynamic specification mining algorithm based on merging states whose future k states are identical. It has been used in several variants in many recent works, e.g., [3], [9], [24], [28], [29], [39]. Below we reformulate the computations of [8] using the framework defined above.

1) Log Properties: To apply the log completeness framework to k-Tails we define a single log-property, $LP^{\triangleright^k}$, for the existential property 'k-directly-follows'. Roughly, the property 'k-directly follows' for a sequence of events $es = \langle e_1, e_2, \ldots, e_k\rangle$ holds in a trace iff the sequence appears somewhere in the trace.

The intuition behind the use of this property is as follows. Consider a model produced by k-Tails for a log, and a new trace whose all subtraces of length $k+1$ appear in the produced model. Applying k-Tails to a new log that consists of the original log and the new trace, produces the same model. The proof for this claim, albeit using a different formulation, can be found in [4].

The definitions of $LP^{\triangleright^k}_{tr}$ and $LP^{\triangleright^k}_{log}$ below follow the semantics of 'k-directly-follows' as an existential property. Formally, we use $D_{tr} = D_{log} = \{0, 1\}$ and define:
$LP(A_{\text{k-Tails}}) = \{LP^{\triangleright^k}\}$ where

$$LP^{\triangleright^k}_{tr}(\sigma, \langle e_1, e_2, \ldots, e_k\rangle) = \begin{cases} 1 & \exists_j \bigwedge\limits_{1 \leq m \leq k} \sigma(j+m-1) = e_m \\ 0 & otherwise \end{cases}$$

$$LP^{\triangleright^k}_{log}(S) = \begin{cases} 1 & 1 \in S \\ 0 & otherwise \end{cases}$$

2) Computing Log Confidence: The probability that an existential property does not hold in $T(M)$ but appears in one of the traces is zero. Thus, we only need to consider the other case, where the sequence of length k does not appear in the traces ($Y^n[es] = 0$) but is possible in the model ($f[es] = 1$). Formally:[1]

$$\begin{aligned}\mathbb{P}[Y^n = f] &= 1 - \mathbb{P}[\exists es.Y^n[es] = 0 \wedge f[es] = 1] \\ &\geq 1 - \sum_{es} \mathbb{P}[Y^n[es] = 0 \wedge f[es] = 1]\end{aligned}$$

$$\begin{aligned}\mathbb{P}[Y^n[es] = 0 \wedge f[es] = 1] &= \begin{cases} \mathbb{P}[Y^n[es] = 0] & f[es] = 1 \\ 0 & f[es] = 0 \end{cases} \\ &= \begin{cases} \prod\limits_{1 \leq i \leq n} \mathbb{P}[Y_i[es] = 0] & f[es] = 1 \\ 0 & f[es] = 0 \end{cases}\end{aligned}$$

We use q_{es} to denote the probability that the existential property for es holds on a random trace from $T(M)$, i.e., that

[1] Since they are fixed, we omit the specific LP and π from the formulas in this section

the sequence $\langle e_1, e_2, \ldots, e_k \rangle$ appears somewhere in the trace. When $f[es] = 1$ we have $q_{es} > 0$ and $\prod_{1 \leq i \leq n} \mathbb{P}[Y_i[es] = 0] = (1 - q_{es})^n$.

Since q_{es} is unknown, we estimate it using the average of the n random variables Y_i

$$\hat{q}_{es} = \sum_{i=1}^{n} \frac{Y_i[es]}{n}$$

and so overall, we have

$$\mathbb{P}[Y^n = f] \geq 1 - \sum_{\{es | \hat{q}_{es} > 0\}} (1 - \hat{q}_{es})^n.$$

B. Synoptic

Synoptic [5] is a dynamic specification mining algorithm based on three temporal invariants of length 2 and a process of refinement/coarsening using counter-example-guided-abstraction-refinement (CEGAR) and a variant of k-Tails.

1) Log Properties: To apply the framework to Synoptic we define four log-properties, for three invariant properties and for one existential property. The three invariant properties are 'always followed by' (denoted $\rightarrow$), 'always precedes' (denoted $\leftarrow$), and 'never followed by' (denoted $\nrightarrow$). The existential property is '2-directly-follows' (denoted $\rhd_2$). Formally: $LP(A_{\text{Synoptic}}) = \{LP^{\rightarrow}, LP^{\leftarrow}, LP^{\nrightarrow}, LP^{\rhd_2}\}$, and all four Synoptic's log-properties use $D_{tr} = D_{log} = \{0, 1\}$.

The intuition behind the use of these properties is as follows. First, the three invariants are mined by Synoptic and then used during the refinement process; the final model produced by Synoptic is guaranteed to satisfy all three invariants. Second, the existential '2-directly-follows' property corresponds to the initial step in the Synoptic algorithm, which builds a permissive model that accepts all traces. Formal correctness proof for the selection of these properties appear in [4].

The definition of $LP^{\rightarrow} = \langle LP^{\rightarrow}_{tr}, LP^{\rightarrow}_{log} \rangle$ is based on its semantics. $LP^{\rightarrow}_{tr}$ takes two events as input and outputs 1 iff every occurrence of the first is followed by an occurrence of the second. $LP^{\rightarrow}_{log}(S)$ takes the result of applying $LP^{\rightarrow}_{tr}$ to all traces and outputs 1 iff all traces satisfy the invariant. Formally:

$$LP^{\rightarrow}_{tr}(\sigma, \langle e_1, e_2 \rangle) = \begin{cases} 1 & \forall_k \sigma(k) = e_1 \Rightarrow \exists_{m>k} \sigma(m) = e_2 \\ 0 & otherwise \end{cases}$$

$$LP^{\rightarrow}_{log}(S) = \begin{cases} 1 & 0 \notin S \\ 0 & otherwise \end{cases}$$

The definition of the two other invariants is similar: $LP^{\leftarrow}_{tr}$ takes two events as input and outputs 1 iff every occurrence of the second is preceded by an occurrence of the first; $LP^{\nrightarrow}_{tr}$ takes two events as input and outputs 1 iff no occurrence of the first is followed by an occurrence of the second. Formally:

$$LP^{\leftarrow}_{tr}(\sigma, \langle e_1, e_2 \rangle) = \begin{cases} 1 & \forall_k \sigma(k) = e_2 \Rightarrow \exists_{m<k} \sigma(m) = e_1 \\ 0 & otherwise \end{cases}$$

$$LP^{\nrightarrow}_{tr}(\sigma, \langle e_1, e_2 \rangle) = \begin{cases} 1 & \forall_k \sigma(k) = e_1 \Rightarrow \neg\exists_{m>k} \sigma(m) = e_2 \\ 0 & otherwise \end{cases}$$

All three log-properties are invariants, so they have the same LP_{log} function, that is $LP^{\rightarrow}_{log} = LP^{\leftarrow}_{log} = LP^{\nrightarrow}_{log}$.

Finally, the log-property $LP^{\rhd_2}$, relating to the existential property '2-directly-follows', is a special case of the 'k-directly follows' property defined above for k-Tails (see Sect. IV-A1), for $k = 2$.

2) Computing Log Confidence: We estimate the probability of log completeness with regard to Synoptic by taking the minimum of the probabilities of log completeness for each property alone.

The three invariant properties. We use es to denote the sequence $\langle e_1, e_2 \rangle$.

For the three invariant properties, the computation is the same. The probability that the invariant holds in $T(M)$ but is violated in one of the traces is zero. Thus, we only need to consider the other case, where the invariant holds in all traces that we have seen ($Y^n[es] = 1$) but not in $T(M)$ ($f[es] = 0$). Formally:

$$\begin{aligned} \mathbb{P}[Y^n = f] &= 1 - \mathbb{P}[\exists es. Y^n[es] = 1 \wedge f[es] = 0] \\ &\geq 1 - \sum_{es} \mathbb{P}[Y^n[es] = 1 \wedge f[es] = 0] \end{aligned}$$

$$\begin{aligned} \mathbb{P}[Y^n[es] = 1 \wedge f[es] = 0] &= \begin{cases} \mathbb{P}[Y^n[es] = 1] & f[es] = 0 \\ 0 & f[es] = 1 \end{cases} \\ &= \begin{cases} \prod_{1 \leq i \leq n} \mathbb{P}[Y_i[es] = 1] & f[es] = 0 \\ 0 & f[es] = 1 \end{cases} \end{aligned}$$

We use q_{es} to denote the probability that the invariant holds for es on a random trace from $T(M)$. When $f[es] = 0$ we have $q_{es} < 1$ and $\prod_{1 \leq i \leq n} \mathbb{P}[Y_i[es] = 1] = q^n_{es}$.

Since q_{es} is unknown we estimate it using the average of the n random variables Y_i

$$\hat{q}_{es} = \sum_{i=1}^{n} \frac{Y_i[es]}{n} \tag{1}$$

and overall, for each of the three invariant properties we have

$$\mathbb{P}[Y^n = f] \geq 1 - \sum_{\{es | \hat{q}_{es} < 1\}} (\hat{q}_{es})^n. \tag{2}$$

The existential property. Computing confidence for the existential property '2-directly-follows' is again a special case of the computation of 'k-directly follows' we have shown above in Sect. IV-A2, so we do not repeat it here.

Finally, we consider the log's confidence with regard to the Synoptic algorithm as a whole to be the minimum of the four probabilities of the log completeness properties.

C. Mining Scenario-Based Triggers & Effects

Mining scenario-based triggers and effects (mining t/e) was presented in [25]. Roughly, given a trigger scenario (a sequence of events), the miner looks for all effect scenarios (sequences of events) such that for each, whenever the trigger occurs in a trace (in the specified events order but possibly with other events interleaved), eventually the effect occurs in this trace

(in the specified events order but possibly with other events interleaved). Below we use the notation from [25].

Given a trigger scenario tg and an effect scenario es, $pos(tg{+\!\!+}es)$ is the set of all positive witnesses of the combined scenario $tg{+\!\!+}es$, i.e., all cases where an occurrence of tg is eventually followed by an occurrence of es; and $neg(tg{+\!\!+}es)$ is the set of all negative witnesses of the combined scenario $tg{+\!\!+}es$, i.e., all cases where an occurrence of tg is not followed by an occurrence of es. Formal definitions appear in [25].

We consider the case where the input consists of a trigger and the miner looks for effects. The other case is symmetric.

1) Log Properties: We define a single log-property, corresponding to the following: given a trigger and a candidate effect, is it true that whenever the trigger occurs, eventually the effect occurs too? We denote this trigger/effect property by t/e: $LP(A_{\text{Trigger/Effect}}) = \{LP^{t/e}\}$.

The intuition behind the use of this property is that it is equivalent to the property which the algorithm looks for: log completeness with regard to this property entails that the mining algorithm results are indeed correct and complete with regard to the true model.

The definition of $LP^{t/e} = \langle LP^{t/e}_{tr}, LP^{t/e}_{log}\rangle$ is based on its semantics. $LP^{t/e}_{tr}$ takes a trace and a candidate effect as input and outputs 1 (true) iff the trace has at least one positive witness and no negative witnesses for the combined trigger effect, 0 (false) iff the trace has at least one negative witness for the combined trigger effect, and -1 (unknown) if the trigger never occurs in the trace. The domains for the log-property $\{LP^{t/e}\}$ are three valued: $D_{tr} = D_{log} = \{1, 0, -1\}$.

$LP^{t/e}_{log}(S)$ takes the result of applying $LP^{t/e}_{tr}$ to all traces and outputs 1 iff it returned 1 for at least one trace and returned 0 for no trace. Formally:

$$LP^{t/e}_{tr}(\sigma, es) = \begin{cases} 1 & |pos(tg{+\!\!+}es)| > 0 \wedge |neg(tg{+\!\!+}es)| = 0 \\ 0 & |neg(tg{+\!\!+}es)| > 0 \\ -1 & otherwise \end{cases}$$

$$LP^{t/e}_{log}(S) = \begin{cases} 1 & 1 \in S \wedge 0 \notin S \\ 0 & 0 \in S \\ -1 & otherwise\ (S = \{-1\}) \end{cases}$$

2) Computing Log Confidence: We use es to denote the sequence $\langle e_1, e_2, \ldots\rangle$; the length of the effect we are looking for is unbounded.

The computation considers the possible cases where the log and the model do not agree. First, the case where the trigger does not occur in the log ($Y^n[es] = -1$) although it is possible in $T(M)$ ($f[es] \neq -1$). Second, the case where the trigger occurs in the log with no negative witnesses ($Y^n[es] = 1$) although the combined trigger effect is not true $T(M)$ ($f[es] = 0$). (Other cases are impossible, e.g., the case where we see a negative witness in the log although the combined trigger effect is true in $T(M)$). Formally:

$$\begin{aligned} \mathbb{P}[Y^n = f] &= 1 - (\mathbb{P}[\exists es Y^n[es] \neq f[es]]) \\ &\geq 1 - \big(\textstyle\sum_{es}\mathbb{P}[Y^n[es] = -1 \wedge f[es] \neq -1] \\ &\quad + \textstyle\sum_{es}\mathbb{P}[Y^n[es] = 1 \wedge f[es] = 0]\big) \end{aligned}$$

We compute each of the two probabilities above as follows. We use $q_{tg{+\!\!+}es}$ and q_{tg} to denote the probability that a random trace from $T(M)$ has only positive witnesses of $tg{+\!\!+}es$ and tg respectively. For the first probability, we have

$$\begin{aligned} \mathbb{P}[Y^n[es] = -1 \wedge f[es] \neq -1] &= \begin{cases} \mathbb{P}[Y^n[es] = -1] & f[es] \neq -1 \\ 0 & otherwise \end{cases} \\ &= \begin{cases} \mathbb{P}[Y^n[es] = -1] & q_{tg} > 0 \\ 0 & otherwise \end{cases} \end{aligned}$$

When $f[es] \neq -1$ we have $q_{tg} > 0$ because tg is possible in the model. In this case, by definition of $LP^{t/e}_{log}(S)$, we have the probability that the trigger will not occur in any of the n traces:

$$\mathbb{P}[Y^n[es] = -1] = (1 - q_{tg})^n.$$

For the second probability, we have

$$\mathbb{P}[Y^n[es] = 1 \wedge f[es] = 0] = \begin{cases} \mathbb{P}[Y^n[es] = 1] & f[es] = 0 \\ 0 & otherwise \end{cases}$$

When $f[es] = 0$ we have $q_{tg} \neq q_{tg{+\!\!+}es}$ because $tg{+\!\!+}es$ does not hold in the model. In this case, by definition of $LP^{t/e}_{log}(S)$, we have to consider the case where in the n traces we have seen the trigger at least once and we have not seen any negative witnesses:

$$\mathbb{P}[Y^n[es] = 1] = (1 - (1 - q_{tg})^n) \times (1 - (q_{tg} - q_{tg{+\!\!+}es}))^n$$

Finally, since q_{tg} and $q_{tg{+\!\!+}es}$ are unknown we estimate them using the n random variables Y_i

$$\hat{q}_{tg} = \sum_{Y_i[es] \in \{1,0\}} \frac{1}{n} \quad , \quad \hat{q}_{tg{+\!\!+}es} = \sum_{Y_i[es]=1} \frac{1}{n}$$

and so overall we have

$$\begin{aligned} \mathbb{P}[Y^n = f] \geq 1 - \big(&\textstyle\sum_{\hat{q}_{tg} > 0}(1 - \hat{q}_{tg})^n \\ &+ \textstyle\sum_{\hat{q}_{tg} \neq \hat{q}_{tg{+\!\!+}es}} (1 - (1 - \hat{q}_{tg})^n) \times (1 - (\hat{q}_{tg} - \hat{q}_{tg{+\!\!+}es}))^n\big). \end{aligned}$$

D. Implementation

We have implemented the computation of log confidence for the three dynamic specification algorithms. For each of the three algorithms, the implementation gets as input a log (a set of traces) and algorithm specific parameters (none for Synoptic, k for k-Tails, and the trigger or effect sequence for scenario-based trigger and effect). Note that the confidence computation does not need to run the mining algorithm.

Given a log l and an algorithm A, computing the log's confidence starts with separately computing its confidence for each of the log-properties in $LP(A)$. For each log-property LP, for each trace $\sigma \in l$, and for each sequence es, we compute $LP_{tr}(\sigma, es)$. The computed values are used as input for the computation of the log's confidence with regard to LP. The reported log's confidence of l with regard to A, is the minimum of all the confidences for the log-properties.

event	close	closeEntry	init	putNextEntry	write
close	-	0.43 (-0.00)	0.00 (-0.00)	0.29 (-0.00)	0.43 (-0.00)
closeEntry	0.57 (-0.02)	-	0.00 (-0.00)	0.57 (-0.02)	0.57 (-0.02)
init	1.00 (-0.00)	1.00 (-0.00)	-	1.00 (-0.00)	1.00 (-0.00)
putNextEntry	0.71 (-0.10)	0.57 (-0.02)	0.00 (-0.00)	-	1.00 (-0.00)
write	0.57 (-0.02)	0.57 (-0.02)	0.00 (-0.00)	0.29 (-0.00)	-

TABLE I: Example log confidence computation with regard to the invariant property 'always precedes' ($LP^{\leftarrow}$), for the log of 7 traces shown in Fig. 2. The computed confidence for this property is 0.78. See Sect. V.

V. Example Computation

We demonstrate log confidence computation on our running example model and 7 randomly generated traces from this model, shown in Fig. 2. For these traces, Table I shows the computation of log confidence with regard to the invariant property 'always precedes' ($LP^{\leftarrow}$, as used in the confidence computation for Synoptic, see Sect. IV-B).

The table cell i, j corresponds to the invariant 'i always precedes j', e.g., the table cell in the row of `closeEntry` and column of `close` corresponds to the property '`closeEntry` always precedes `close`'. The value in table cell i, j is $\hat{q}_{\langle i,j \rangle}$ from Equ. 1. For example, the value $\hat{q}_{\texttt{closeEntry;close}} = 0.57$ is the probability to have an instance of `closeEntry;close` with 'always precedes' in a random trace, given the log that we have (the invariant holds in 4 of the 7 traces (tr_1,tr_2,tr_3, and tr_7), so $4/7 = 0.57$ is the probability that `closeEntry` always precedes `close`). Since the log has 7 traces, the negative contribution to the accumulating confidence for this property is $(0.57)^7 = 0.02$ (see Equ. 2) (that's the probability that in a random log of size 7, this invariant will hold).

The overall confidence for the 'always precedes' property is computed by assigning all the numbers from the table to the $\hat{q}_{es}$s in Equ. 2 (we omit the zeros from the formula):
$1 - (0.02 + 0.1 + 0.02 + 0.02 + 0.02 + 0.02 + 0.02) = 0.78$.

VI. Evaluation

The research questions guiding our evaluation are:

- RQ1 Is the representation of the three specification mining algorithms using LPs sound?
- RQ2 Can log confidence be efficiently computed and serve as an effective proxy for true log completeness?

A. Models Used in Evaluation

In the evaluation we used 24 finite-state automaton models, taken from 9 publicly available previously published works and reports: [12], [19], [23], [28], [32], [34]–[36] and [40]. The models varied in size and complexity: the alphabet size ranged from 7 to 42 (mean 14.42), the number of states ranged from 5 to 24 (mean 12.62), and the number of transitions ranged from 15 to 209 (mean 37.58).

All logs, models, and implementation code described in this paper are available for inspection and reproduction together with documentation from [1].

B. RQ1: Soundness of Representation

1) Methodology: To evaluate the soundness of the representation of the three specification mining algorithms using LPs we use two definitions: LP-equivalence and A-equivalence.

First, roughly, given a log-property LP, two logs are LP-equivalent if their results agree on all sequences. Formally:

Definition 5 (LP-equivalence)**.** For a log-property LP, two logs l_1, l_2 are LP-equivalent iff $\forall es \in \Sigma^i$

$$LP_{log}(\{LP_{tr}(\sigma, es) | \sigma \in l_1\}) = LP_{log}(\{LP_{tr}(\sigma, es) | \sigma \in l_2\}).$$

We trivially lift the above definition to a set of LPs, that is, from each LP alone to the set $LP(A)$.

Second, roughly, given a specification mining algorithm A, two logs are A-equivalent if they agree on the result of the algorithm. Formally:

Definition 6 (A-equivalence)**.** For a specification mining algorithm A, two logs l_1, l_2 are A-equivalent iff $A(l_1) = A(l_2)$.

Finally, we say that the LP representation of an algorithm A, $LP(A)$, is sound, iff $LP(A)$-equivalence implies A-equivalence.

2) Experiment Design: For each model we used the following experiment protocol. First, we generated traces from the model using a trace generator. Second, for each algorithm, we found the minimal sub log (in some arbitrary order of traces) which is $LP(A)$-equivalent to the entire log. Third, we ran the specification mining algorithm A on the two logs and checked whether the output is the same.

For each of the 24 models, and for each specification mining algorithm, k-Tails, Synoptic, and mining t/e, we ran the experiment three times. In all experiments we used the trace generator from [28] with path coverage (but high state coverage for some of the models because the generator ran out of memory when computing path coverage for these models).

3) Results: In all executions, i.e., for all models and for all algorithms, the model generated from the sub log was *identical* to the one generated from the entire log.

To answer [RQ1], we have strong evidence for the soundness of the LPs representation, for all three algorithms.

C. RQ2: Effectiveness of Log Confidence

1) Methodology: To evaluate the effectiveness of log confidence we use two key measures, reliability and redundancy.

For a fixed algorithm A, we define the reliability of a log to be 1 if the log is complete with regard to A and 0 otherwise. For a set of logs L, a mean reliability close to 1 hints that most of the logs are complete. For a fixed algorithm A, we define the redundancy of a log to measure how close is it to its minimal prefix log which is complete (assuming an arbitrary fixed order of traces). For a set of logs L, a mean redundancy close to 0 and a low standard deviation hint that the logs do not include much redundant traces. Formally:

Definition 7 (reliability)**.** For an algorithm A, the reliability of a log l is $rel(l) = \begin{cases} 1 & l \text{ is complete with regard to } A \\ 0 & otherwise \end{cases}$.

Definition 8 (redundancy)**.** For an algorithm A, given a log l (in a fixed arbitrary order), let $i_{min}(l)$ be the minimal index of traces in l such that the set of traces $\sigma_1, \sigma_2, \ldots, \sigma_{i_{min}} \in l$ is complete with regard to A. The redundancy of a log l is $red(l) = 1 - \frac{i_{min}(l)}{|l|}$.

Example 5. Recall the log and model of our running example shown in Fig. 2, and the results of computing its confidence with regard to the 'always precedes' property, as shown in Table I. This log's reliability with regard to this property is $rel(l) = 1$. Since it reaches completeness for this property already after the 4th trace and we have 7 traces, its redundancy with regard to this property is $red(l) = 1 - 4/7 = 0.43$.

Note that to calculate reliability and redundancy, as in the above example, one must know the system from which the traces were extracted, so that she can calculate a true value for each property. This is typically unknown in a real-world setting, but it is known in our controlled evaluation setting.

In order to answer [RQ2], we are interested in the performance of the confidence computation, in the values of the reliability and their correlation with the confidence values, and in the redundancy values, across the 24 models, for each of the three algorithms.

2) Experiment Design: For each model we used the following experiment protocol. First, we generated traces from the model using a trace generator. Second, we created an initial log by randomly selecting a minimal number of traces. Third, for several fixed thresholds, we iteratively computed the current log's confidence and added a trace to it; adding traces to the current log until we reached the fixed confidence threshold (or we ran out of traces to add). Finally, we computed the reliability and redundancy of the final log.

For each model and for each algorithm, we repeated the above protocol 200 times and computed the mean of reliability and redundancy for the sets of 200 logs.

In all experiments we used the trace generator of [28] with path coverage (high state coverage for some of the models because the generator ran out of memory when computing path coverage for them), a minimal number of 10 traces, and a series of fixed confidence thresholds, from 0.20 to 0.95; i.e., for each threshold th, we started with an initial log by randomly selecting 10 traces, and continued the addition of traces to the log until the probability that it is complete was at least th (or we ran out of traces to add). We checked true completeness using a model-checker, i.e., by expressing the relevant log-properties in temporal logic and verifying them against the model.

We used $k = 2$ for k-Tails, and manually selected a trigger of length 2 or 3 for the triggers and effects algorithm.

3) Results: **Performance.** All experiments were executed on an ordinary laptop computer, Intel i7 CPU 3.0GHz, 8GB RAM with Windows 7 64-bit OS, Java 1.7.0_09 64-bit. For all models, in all our experiments, confidence computation never exceeded 15 milliseconds. This shows that the log confidence computation for the three specification mining algorithms we deal with is fast. It is not surprising as the computation is, by definition, linear in the number of traces in the log.

Reliability. Fig. 3 shows the reliability results across the 24 models, for increasing confidence thresholds, for each of the three algorithms. The boxplots show the median reliability, the 25th and 75th percentile with range 3/2 for whiskers.

For all three algorithms, the boxplots show that in general, the reliability increases as the confidence threshold increases, and the variance in reliability across the different models decreases as the confidence threshold increases. Specifically, when the confidence threshold is 0.95, the reliability is very high, and its variance across the different models is very low.

The boxplots also show that the reliability is always greater or equal to the confidence threshold. This is a result of the conservative nature of our confidence computation; it is much more likely to underestimate completeness than to overestimate it. It is also a result of our experiment design: we stop adding traces to the log once its confidence pass the given threshold, i.e, the actual confidence is higher than the threshold used and shown in the figure (the lower the threshold, the higher the possible difference).

The Spearman's rank correlation between confidence and reliability was $\rho = .68$ ($p < .05$), $\rho = .70$ ($p < .05$), and $\rho = .52$ ($p < .05$), for k-Tails, Synoptic, and triggers and effects, respectively. These values are considered to express strong correlation [10, p. 140]. A power analysis for the correlation tests shows that our sample size of $n = 120$ is much above the minimal size required with significance level $\alpha = .05$, power = .8, and our resulting correlation coefficients.

Redundancy. Fig. 4 shows the redundancy results across the 24 models, for increasing confidence thresholds, for each of the three mining algorithms. For all algorithms, the boxplots show that in general, as expected, the redundancy increases as the confidence threshold increases. When confidence threshold is .95, the maximal redundancy is of about .74 and the maximal median redundancy is about .63. The worst case for the three algorithms was reaching the .95 threshold with a log that is about 4 times longer than its minimal complete sublog.

Log sizes differed much between the different models and across the different confidence thresholds, roughly ranging from 20 to 2000 traces per log. Some models required many more traces than others to reach high confidence (across all three algorithms). The results for the individual models are available from [1].

We observe that the variance between the different models, for both reliability and redundancy, seems higher for mining t/e than for the other two algorithms. This may be viewed as

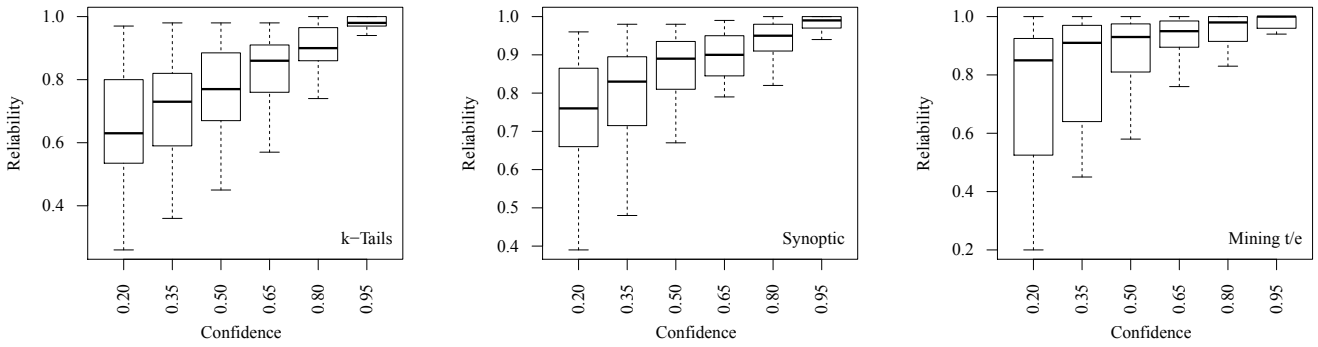


Fig. 3: Reliability results for the three specification mining algorithms. See Sect. VI-C.

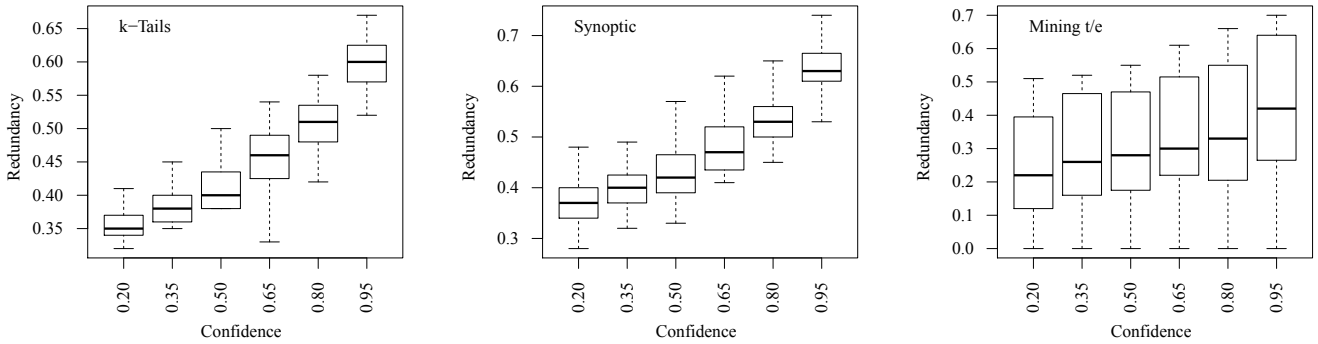


Fig. 4: Redundancy results for the three specification mining algorithms. See Sect. VI-C.

a weakness. However, the median values for mining t/e, are higher for reliability and lower for redundancy, compared to the other two algorithms, which may be viewed as a strength. Note though that the kind of candidate specification mined by mining t/e is very different from the kind mined by the other two algorithms, hence they are not comparable.

To answer [RQ2], we have strong evidence that log confidence can be efficiently computed and is an effective proxy for true completeness for the three specification mining algorithms.

D. Threats to Validity

We discuss threats to the validity of our results. First, the selection of models in our evaluation may not represent typical systems. To mitigate this, we used 24 publicly available models taken from 9 previous works (see Sect. VI-A). Yet, we do not know if these are representative of real-world systems.

Second, in our evaluation we used a publicly available trace generator [28] with path coverage (when the model was too complex for the trace generator to handle, we used state coverage). It is possible that one may get different results if a different trace generator or a different coverage criterion are used. In a real-world setting, when the model is unknown, code coverage methods may be used.

Third, the selection and order of traces in the log affect the point where the analysis may reach the confidence threshold and thus affect the point used to compute redundancy. We mitigated this by using randomization in the selection of traces and in the order in which they were analyzed, and by repeating all evaluation experiments 200 times.

Fourth, we have evaluated the soundness of the LPs empirically against specific implementations of the three algorithms. All have variants, which may produce slightly different outputs, e.g., Synoptic chooses the order in which invariants are used in the CEGAR process arbitrarily. Thus, when we write $A(l)$ in Sect. VI-B, we refer to a specific implementation only.

Finally, our work presents a general framework for log completeness and confidence but the evaluation is limited to three algorithms. One may get different results when the framework is applied to other algorithms. Still the results provide some evidence for the generalizability of the framework.

VII. Discussion

We now discuss some important features, design decisions, and several limitations of our present work. An important feature of our log confidence computation is that it is not monotonic but converges to 1 when the size of the log approaches infinity (for all three algorithms). On the one hand, as traces are added to a log, its confidence may increase or decrease; in the absence of additional information, e.g., about the order in which traces are produced, a new trace may

introduce new information (e.g., invalidate an invariant, reveal a new k-length sequence), which leads us to revise our previous estimations. Thus, confidence is not necessarily monotonic. Still, in contrast, it is important to note that our notion of log completeness is monotonic, i.e., by definition, any extension of a complete log is complete. On the other hand, despite the non monotonicity, the expected confidence converges to 1 when the size of the log approaches infinity (for the three algorithms). For example, for k-Tails, we have

$$\lim_{n \to +\infty} \mathbb{E}\left[1 - \sum_{\{es | \hat{q}_{es} > 0\}} (1 - \hat{q}_{es})^n\right] = 1.$$

Thus, as the size of the log grows, the expected change in confidence that an additional trace may cause approaches zero. The proof for this claim can be found in [1].

In the presence of multiple LPs in $LP(A)$, we have chosen to define confidence as the minimum of confidences over all LPs. We consider this to be a preliminary, conservative choice, which, in the tradeoff between high reliability and low redundancy, favors the former over the latter. Alternatively, one may choose other less conservative yet perhaps more robust aggregate functions, e.g., the harmonic mean, or some weighted average in case some LPs are considered more important than others. We leave the investigation of these alternatives, per algorithm, to future work.

An important assumption underlying our framework is that the traces are randomly and independently chosen from $T(M)$. This assumption may not always hold, e.g., if traces depend on running tests that were generated according to some strategy. We note, however, that our work is meant to be used to evaluate the confidence one may have in the results of applying mining algorithms to logs of execution traces. This should not be confused with evaluating the quality of a test suite.

Finally, an apparent limitation relates to the size of the log. For logs with only a few traces, confidence results are very sensitive and fluctuate much, so they are practically useless. In practice however this is typically not a problem because real-world logs consist of many traces. Moreover, as stated above, as the size of the log grows, the expected fluctuations approach zero.

VIII. Related Work

Dynamic Specification Mining. Much literature deals with looking for candidate specifications in execution traces (e.g., [5], [13]–[15], [17], [18], [21], [22], [26], [27], [29]–[31], [37], [38], [41], [42]). The works differ in the kinds of traces they take as input and in the kinds of candidate specification they present as output.

In recent work, Beschastnikh et al. [4] have presented InvariMint, a framework for declarative specification of FSM-inference dynamic log analysis algorithms. Our work may be viewed as building on InvariMint's presentation of model inference algorithms using declarative properties, as a key to its ability to define log completeness for k-Tails and Synoptic. Formally, however, we define properties in a different way. While InvariMint uses parameterized FSM and evaluation functions to describe property types, and applies composition functions to combine them, our framework describes properties using the functions LP_{tr} and LP_{log}, which enable the computation of probabilities. It may be possible to define our notion of log completeness on top of the InvariMint framework. We leave this for future work.

Addressing Completeness. Dallmeier et al. [12] consider dynamic specification mining and ask: what makes us believe that we have seen sufficiently many executions? While it seems that we ask a similar question, our work is fundamentally different. In [12], a partial set of traces is heuristically enriched in order to explore previously unobserved aspects of the execution space. In contrast, we consider a black box setting and provide a formal probabilistic measure to the notion of 'sufficiently many traces'. This allows engineers to assess the completeness of the traces they have, relative to the dynamic specification mining algorithm of their choice.

Hee et al. [33] presented a probabilistic approach to log completeness of the α-algorithm [2], which mines Petri nets and is used in the field of process mining. Our work extends the work of Hee et al. significantly: we present an algorithm-independent framework for log completeness and demonstrate its application to three algorithms from the dynamic specification mining literature, k-Tails, Synoptic, and mining t/e.

Apart from our own recent preliminary work [8], where we have presented a notion of confidence computation for k-Tails, with preliminary evaluation, to the best of our knowledge, no other work has considered the question of estimating log completeness with regard to these three algorithms specifically and with regard to dynamic specification mining in general.

IX. Conclusion

In this paper we addressed the question of analyzing too few or too many traces, in the context of dynamic specification mining, by presenting a novel, black box, probabilistic framework based on a notion of log completeness. We applied the framework to three dynamic specification mining algorithms from the literature. Our evaluation over 24 models from the literature provided evidence for the effectiveness and usefulness of our work.

Given large logs as input, whose analysis may require too much time and memory to the extent that make it impractical, our work can be used to analyze only a fraction of the log yet to construct a candidate specification which is with high confidence identical to the specification one could build from the complete log. The work is part of our larger project on investigating ways to scale up specification mining and other log analysis algorithms in face of long and complex logs, see [7].

Acknowledgements. We thank the anonymous reviewers for their helpful comments. This work has been partly supported by Len Blavatnik and the Blavatnik Family Foundation.

REFERENCES

[1] Supporting materials website. http://smlab.cs.tau.ac.il/logcompleteness.

[2] Wil M. P. van der Aalst, T. Weijters, and L. Maruster. Workflow mining: Discovering process models from event logs. *IEEE Trans. Knowl. Data Eng.*, 16(9):1128–1142, 2004.

[3] M. Acharya, T. Xie, J. Pei, and J. Xu. Mining API patterns as partial orders from source code: from usage scenarios to specifications. In *ESEC/SIGSOFT FSE*, pages 25–34, 2007.

[4] I. Beschastnikh, Y. Brun, J. Abrahamson, M. D. Ernst, and A. Krishnamurthy. Unifying FSM-inference algorithms through declarative specification. In *ICSE*, pages 252–261, 2013.

[5] I. Beschastnikh, Y. Brun, S. Schneider, M. Sloan, and M. D. Ernst. Leveraging existing instrumentation to automatically infer invariant-constrained models. In T. Gyimóthy and A. Zeller, editors, *SIGSOFT FSE*, pages 267–277. ACM, 2011.

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

[7] N. Busany and S. Maoz. Behavioral Log Analysis with Statistical Guarantees. In *SIGSOFT FSE*, 2015.

[8] H. Cohen and S. Maoz. The Confidence in our k-Tails. In *ASE*, 2014.

[9] J. E. Cook and A. L. Wolf. Discovering models of software processes from event-based data. *ACM Trans. Softw. Eng. Methodol.*, 7(3):215–249, 1998.

[10] G. W. Corder and D. I. Foreman. *Nonparametric Statistics: A Step-by-Step Approach*. Wiley, 2nd edition, 2014.

[11] CrossFTP Server. http://sourceforge.net/projects/crossftpserver/.

[12] 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.

[13] F. C. de Sousa, N. C. Mendonça, S. Uchitel, and J. Kramer. Detecting implied scenarios from execution traces. In *WCRE*, pages 50–59, 2007.

[14] M. El-Ramly, E. Stroulia, and P. G. Sorenson. From run-time behavior to usage scenarios: an interaction-pattern mining approach. In *KDD*, pages 315–324. ACM, 2002.

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

[16] D. Fahland, D. Lo, and S. Maoz. Mining Branching LSCs: CrossFTP, Columba traces and results. 3TU.DataCentrum, 2013. doi:10.4121/uuid:aa7db920-aae6-4750-8975-cb739262f432.

[17] D. Fahland, D. Lo, and S. Maoz. Mining branching-time scenarios. In *ASE*, pages 443–453. IEEE, 2013.

[18] M. Gabel and Z. Su. Online inference and enforcement of temporal properties. In *ICSE*, pages 15–24, 2010.

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

[20] I. Krka, Y. Brun, and N. Medvidovic. Automatic Mining of Specifications from Invocation Traces and Method Invariants. In *SIGSOFT FSE*. ACM, 2014.

[21] S. Kumar, S.-C. Khoo, A. Roychoudhury, and D. Lo. Mining message sequence graphs. In *ICSE*, pages 91–100, 2011.

[22] C. Lee, F. Chen, and G. Rosu. Mining parametric specifications. In *ICSE*, pages 591–600, 2011.

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

[24] D. Lo and S.-C. Khoo. SMArTIC: towards building an accurate, robust and scalable specification miner. In *SIGSOFT FSE*, pages 265–275, 2006.

[25] D. Lo and S. Maoz. Mining scenario-based triggers and effects. In *ASE*, pages 109–118. IEEE, 2008.

[26] D. Lo and S. Maoz. Scenario-based and value-based specification mining: better together. In *ASE*, pages 387–396, 2010.

[27] D. Lo, S. Maoz, and S.-C. Khoo. Mining modal scenario-based specifications from execution traces of reactive systems. In R. E. K. Stirewalt, A. Egyed, and B. Fischer, editors, *ASE*, pages 465–468. ACM, 2007.

[28] 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.

[29] D. Lorenzoli, L. Mariani, and M. Pezzè. Automatic generation of software behavioral models. In W. Schäfer, M. B. Dwyer, and V. Gruhn, editors, *ICSE*, pages 501–510. ACM, 2008.

[30] L. Mariani, S. Papagiannakis, and M. Pezzè. Compatibility and regression testing of COTS-component-based software. In *ICSE*, pages 85–95. IEEE Computer Society, 2007.

[31] L. Mariani and M. Pezzè. Behavior capture and test: Automated analysis of component integration. In *ICECCS*, pages 292–301, 2005.

[32] 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.

[33] Kees M. van Hee, Z. Liu, and N. Sidorova. Is my event log complete? - a probabilistic approach to process mining. In *RCIS*, pages 1–7. IEEE, 2011.

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

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

[36] M. Pradel, P. Bichsel, and T. R. Gross. A framework for the evaluation of specification miners based on finite state machines. In *ICSM*, pages 1–10. IEEE Computer Society, 2010.

[37] M. Pradel and T. R. Gross. Automatic generation of object usage specifications from large method traces. In *ASE*, pages 371–382. IEEE Computer Society, 2009.

[38] J. Quante and R. Koschke. Dynamic protocol recovery. In *WCRE*, pages 219–228, 2007.

[39] S. P. Reiss and M. Renieris. Encoding program executions. In H. A. Müller, M. J. Harrold, and W. Schäfer, editors, *ICSE*, pages 221–230, 2001.

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

[41] N. Walkinshaw and K. Bogdanov. Inferring finite-state models with temporal constraints. In *ASE*, pages 248–257. IEEE, 2008.

[42] J. Yang, D. Evans, D. Bhardwaj, T. Bhat, and M. Das. Perracotta: mining temporal API rules from imperfect traces. In *ICSE*, pages 282–291, 2006.