2011 16th IEEE International Conference on Engineering of Complex Computer Systems

Towards Succinctness in Mining Scenario-Based Specifications

David Lo
School of Information Systems
Singapore Management University
Email: davidlo@smu.edu.sg

Shahar Maoz
Dept. of Computer Science 3 (Software Engineering)
RWTH Aachen University, Aachen, Germany
Email: maoz@se-rwth.de

***Abstract*—Specification mining methods are used to extract candidate specifications from system execution traces. A major challenge for specification mining is succinctness. That is, in addition to the soundness, completeness, and scalable performance of the specification mining method, one is interested in producing a succinct result, which conveys a lot of information about the system under investigation but uses a short, machine and human-readable representation.**

In this paper we address the succinctness challenge in the context of scenario-based specification mining, whose target formalism is live sequence charts (LSC), an expressive extension of classical sequence diagrams. We do this by adapting three classical notions: a definition of an equivalence relation over LSCs, a definition of a redundancy and inclusion relation based on isomorphic embeddings among LSCs, and a delta-discriminative measure based on an information gain metric on a sorted set of LSCs. These are applied on top of the commonly used statistical metrics of support and confidence.

A number of case studies show the utility of our approach towards succinct mined specifications.

I. Introduction

Specification mining, the dynamic analysis process of extracting models from software execution traces in order to aid in program comprehension, testing, and formal verification tasks in the absence of complete and up-to-date documented specifications, has attracted many and diverse research efforts in recent years (see, e.g., [4], [27], [28]). The underlying common goal of these works is to extract a set of invariants and present them to the user. In the literature, extracted invariants vary from boolean expressions capturing a relationship between two variables in a particular program point, to frequent patterns of usage behavior, to various automata and temporal rules. The extracted candidate invariants are used to support comprehension, testing, and formal verification tasks (see, e.g., [4], [7], [8], [31], [33], [32]).

In addition to the soundness, completeness, and scalable performance of the specification mining method, one is interested in producing a succinct result, which conveys a lot of information about the system under investigation but uses a short, machine and human-readable representation. Thus, an important challenge of specification mining in general is the definition of what constitutes a likely invariant and, in particular, the selection and presentation of the extracted invariants. Obviously, a specification mining framework that returns 'too many' likely invariants is not useful. The need arises to produce succinct mined specifications.

In this work we address the succinctness challenge by adapting three classical notions: equivalence relation, statistical redundancy, and delta-discriminative information gain. We do this in the context of *scenario-based specification mining* [22], [23], [26].

In scenario-based specification mining, data mining methods are employed to extract statistically significant inter-object scenario-based specifications in the form of Damm and Harel's *live sequence charts* (LSC) [6], a visual formalism that extends classical sequence diagrams with modalities. An LSC is composed of a pre-chart and a main-chart both consisting an ordered sequence of events. An LSC specifies a temporal invariant: whenever the pre-chart events happen in the specified order, eventually the main-chart events must happen in the specified order. LSC has a formal semantics [6], a UML2-compliant variant [14], and a translation into temporal logics [18]. The popularity and intuitive nature of sequence diagrams as a specification language in general, together with the additional unique features of LSC, motivated our choice of target formalism. The choice is supported by previous work on LSC (e.g., [19], [29]), which can be used to visualize, analyze, manipulate, test, and verify mined specifications.

The statistical significance of an LSC in a set of traces is measured using *support* and *confidence* – two metrics commonly used in data mining [11]. The main inputs are a set of traces and thresholds for minimum support and confidence: the first tells the number of times an LSC need to be observed in the traces, the second dictates the likelihood that the pre-chart of the LSC is indeed followed by the main-chart. Only LSCs obeying the minimum support and confidence are deemed significant (see [26]).

Most importantly, to achieve succinctness we introduce three main features:

1) **A single representative of each equivalence class**. We define an equivalence relation between scenarios based on the notion of symbolic lifelines. The symbolic LSC corresponding to all concrete LSCs in an equivalence class is considered a representative of this class. Statistical significance is computed at the level of the symbolic representative LSC.
2) **Statistical redundancy via isomorphic embeddings**.

978-0-7695-4381-9/11 $26.00 © 2011 IEEE
DOI 10.1109/ICECCS.2011.30

We define an inclusion relation between LSCs based on a variant of subgraph isomorphism. When two LSCs have the same statistical significance but one is embedded in the other, the smaller one is considered redundant and is removed from the result.

3) **Delta-discriminative summarization via information gain**. We rank the mined scenarios and then filter out the ones that are not at least delta-discriminative relative to higher ranked scenarios.

The formal definitions for these features are given in Sec. IV.

It is important to note that setting higher support and confidence thresholds may have the effect of reducing the number of significant mined LSCs. However, this would have removed some informative LSCs while keeping others whose contribution to the complete mined specification is minor. Moreover, given a set of mined LSCs, one may suggest to apply synthesis (see, e.g., [13]), and consider the resulting automaton as a succinct representation of the specification. This alternative, however, has several problems: (1) the resulting automaton representation obfuscates the story-like intuitive nature of the original scenarios, and (2) to the best of our knowledge, synthesis techniques for symbolic LSCs (in the context of object-orientation) or for weighted LSCs (with weights based on support and confidence in our case) are not yet available. Thus, summarization methods such as the ones we introduce and evaluate are indeed necessary.

The examples throughout the paper are taken from CrossFTPServer [5], a commercial open source FTP server built on top of Apache FTP server, and from Jeti [2], a full featured open-source instance messaging application. The examples show the utility of our approach in mining summarized non-redundant symbolic scenarios from a set of execution traces. Experimental results appear in Sec. VI.

Finally, the challenge of succinctness in specification mining is not limited to the scenario-based approach. Some work present mining of rules and patterns (e.g., [8], [32], [20]) where the number of rules and patterns could be very large and make it hard for users to investigate the results. In Daikon [9], the number of reported invariants could be too large for human comprehension, and in mining finite-state machines (e.g., [4], [28]), at times the size of the automaton could be too large to be useful. We further discuss these related works and their succinctness challenges in Sec. VII.

Paper organization: Sec. III provides background on LSC and on statistical significance in scenario-based specification mining. Sec. IV formally defines the three main succinctness features used in our work. Sec. V describes our mining framework. Experimental results appear in Sec. VI. Related work is discussed in Sec. VII and Sec. VIII concludes with a discussion and future work directions.

II. Example

We start off with an example of the application of our succinctness techniques to a mined set of LSCs. The example is intentionally very small and is described semi-formally. Formal definitions appear in Secs. III & IV. The application of our work to large, real-world mined sets of LSCs is described in Sec. VI.

Trace 1		Trace 2		Trace 3	
1	$(a_1,a_2,m_1())$	1	$(a_3,a_4,m_1())$	1	$(a_5,a_6,m_1())$
2	$(a_2,b_1,m_2())$	2	$(a_4,b_2,m_2())$	2	$(a_6,b_3,m_2())$
3	$(a_2,b_1,m_3())$	3	$(a_4,b_2,m_3())$	3	$(a_6,b_3,m_3())$
4	$(a_1,a_2,m_1())$	4	$(a_3,a_4,m_1())$	4	$(a_7,b_4,m_2())$
5	$(a_2,b_1,m_2())$	5	$(a_4,b_2,m_2())$	5	$(a_7,b_4,m_3())$
6	$(a_2,b_1,m_3())$	6	$(a_4,b_2,m_3())$	6	$(a_5,a_6,m_1())$
				7	$(a_6,b_3,m_2())$
				8	$(a_6,b_3,m_3())$

TABLE I
EXAMPLE COLLECTION OF 3 TRACES, EACH OF 6-8 EVENTS. a_i AND b_i ARE OBJECT INSTANCES OF CLASS A AND B RESP. m_1, m_2, AND m_3 ARE METHOD SIGNATURES.

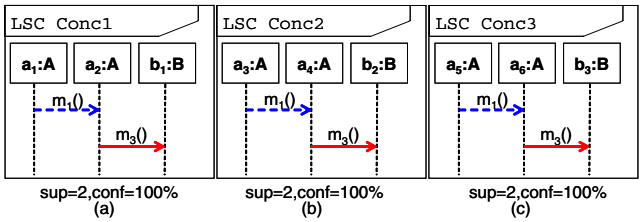

Fig. 1. Some Mined Concrete LSCs

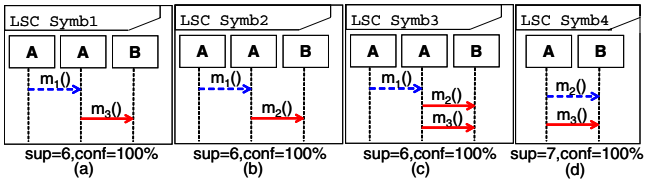

Fig. 2. Some Mined Symbolic LSCs

Fig. 1 shows some mined LSCs from the execution trace shown in Table I. Each mined LSC includes its statistical significance metrics of support and confidence. For example, LSC Conc1 specifies that "whenever object a_1 calls method $m_1()$ of object a_2, a_2 will eventually call method $m_3()$ of object b_1"; its support is 2, meaning it was observed 2 times in the trace; its confidence is 100%, meaning there were no exceptions to this rule: the main-chart always followed the pre-chart. We now show the application of our succinctness techniques to this set of LSCs.

First, LSCs Conc1, Conc2, and Conc3, are very similar: they differ only in the identities of the concrete objects that participate in them. Moreover, in all three LSCs, the participating objects belong to the same set of classes. Our notion of **equivalence class**, which is based on the use of **symbolic LSC lifelines and a binding preserving abstraction**, allows us to identify that these three LSCs belong to the same equivalence class and thus to replace them with a single symbolic LSC. We compute the support and confidence statistics for the symbolic LSC and use it as a representative of the class in the resulting succinct specifications. The symbolic LSC representing the three concrete LSCs is shown in Fig. 2(a).

Second, consider LSCs Symb1, Symb2, and Symb3. LSC Symb3 has the same support and confidence as Symb1 and Symb2. LSC Symb3 is more informative than either Symb1 or Symb2. Our notion of **statistical redundancy** allows us to remove Symb1 and Symb2 as we could embed each of the two LSCs in Symb3. We would only keep LSC Symb3.

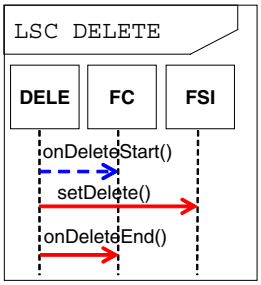

Fig. 3. **[CrossFTP] Mined LSC: Delete**

Third, consider LSC Symb4. Note that Symb4 is not removed based on statistical redundancy. The reason is, Symb4 has a different support than Symb3. However, Symb4 is very similar to Symb3. Our notion of **delta discriminative summarization** allows us to remove Symb4 due to Symb3.

In the process, we could remove a lot of concrete mined LSCs by the three succinctness strategies. In the previous paragraph we notice Symb3 eliminates Symb1, Symb2, and Symb4. Also, Symb1, removes Conc1, Conc2, and Conc3. In summary, using minimum support, confidence, and delta-discriminative thresholds at 2, 100%, and 10% respectively, the application of our succinctness methods will reduce the number of LSCs in this example from 16 to 1.

III. Preliminaries

We give short background on LSC, outline the basic definitions of significance of LSC in a set of traces, and state related properties. We refer the reader to [23], [26] for details.

A. Live sequence charts

We use a restricted subset of live sequence charts (LSC) [6], [14], a formal expressive variant of classical sequence diagrams. A chart includes a set of instance lifelines, representing system's objects, and is divided into two parts, the *pre-chart* ('cold' fragment) and the *main-chart* ('hot' fragment), each specifying an ordered set of method calls between the objects represented by the instance lifelines. A universal LSC specifies a *universal liveness requirement*: for all runs of the system, and for every point during such a run, whenever the sequence of events defined by the pre-chart occurs in the specified order, eventually the sequence of events defined by the main-chart must occur in the specified order. Events not explicitly mentioned in the chart are not restricted in any way to appear or not to appear during the run (including between the events that are mentioned in the chart). Syntactically, instance lifelines are drawn using vertical lines, pre-chart (main-chart) events are colored in blue (red) and drawn using horizontal dashed (solid) lines. For a thorough description of the language and its semantics see [6]. A UML2-compliant variant of LSC using the *modal* profile is defined in [14].

An important feature of LSC is its semantics of *symbolic instances* [30]. Rather than referring to concrete objects, lifelines may be labeled with a class name and marked symbolic, formally representing any object of this class. This allows a designer to take advantage of object-orientation and create more expressive and succinct specifications.

Fig. 3 shows an example LSC. Roughly, the semantics of this LSC is: *"Whenever an object of type DELE calls the method onDeleteStart() of an object of type FtpletContainer, eventually the DELE object must call the method setDelete() of an object of type FtpStatisticsImpl and call the method onDeleteEnd() of the FtpletContainer"*. Note that this is a temporal invariant. Also note the symbolic interpretation of the lifelines.

We denote an LSC by $L(pre, full)$, where *pre* is the pre-chart and $full$ is the complete chart containing pre followed by the main chart. For concrete LSC, the *pre* and $full$ correspond to concrete charts, and for symbolic LSC, the *pre* and $full$ correspond to symbolic charts.

B. Concrete and symbolic events

A concrete event is a triplet (*caller*,*callee*,*signature*) corresponding to caller unique object identifier (obtained in Java using identityHashCode()), callee object identifier, and the signature of the method being called, respectively. In this study, we abstract away method parameters. A trace is a series of concrete events. The input under consideration is a multi-set of traces.

A concrete event could be abstracted to form a symbolic event. A symbolic event is a triplet (*caller*,*callee*,*signature*) corresponding to caller class, callee class, and method signature. While a symbolic event may have one or more corresponding concrete events, a concrete event maps to a single symbolic event. A simple map from a concrete event to its symbolic event is defined as a projection: given a concrete event e, $proj(e)$ returns the symbolic event of e, where the caller and callee objects identifiers are replaced by the names of their classes.

C. Witnesses, support, confidence, and statistical significance

Given a concrete or a symbolic LSC M and a trace T, we are interested in finding statistics denoting the significance of M in T. To do so, we introduce the concepts of positive and negative witnesses. A *positive witness* of a concrete or symbolic LSC $M = L(pre,full)$, is a trace segment satisfying the $full$ chart – by extension the *pre* chart as well, since *pre* is a prefix of $full$. A *negative witness* of M is a positive witness of *pre* which cannot be extended to a positive witness of M (or $full$).

The semantics of LSC (like most formal specification languages used for reactive systems, e.g., LTL [16]) is originally defined over infinite paths. The traces we consider, however, are, of course, finite. To differentiate negative witnesses that occur due to the truncation of the traces, we introduce *weak negative witness*. A *weak negative witness* of M is a positive witness of *pre* which cannot be extended to a positive witness of M (or $full$) due to the end of a trace being reached.

Given a trace T, the *support* of an LSC $M= L(pre, full)$, denoted by $sup(M)$, is simply defined as the number of positive witnesses of M found in T. The *confidence* of an LSC M, denoted by $conf(M)$, measures the likelihood of a sub-trace in T satisfying M's pre-chart to be extended such that M's main-chart is satisfied or the end of the trace is reached. Hence, confidence is expressed as the ratio between the number of

positive witnesses plus weak negative witnesses of the LSC and the number of positive witnesses of the LSC's pre-chart. See [26].

Thus, the support of the chart corresponds to the number of times instances of the pre- and main-chart appear in the trace, and the confidence of the chart corresponds to the likelihood of the pre-chart instance to be followed by a main-chart instance in the trace. The support metric is used to limit the extraction to commonly observed interactions, while the confidence metric restricts mining to such pre-charts that are followed by a particular main-chart with high likelihood. We refer to charts satisfying the minimum support threshold (*m_sup*) as being *frequent*. Similarly, we refer to charts satisfying the minimum confidence threshold (*min_conf*) as being *confident*. A chart that satisfies both thresholds is referred to as being *significant*.

D. Monotonicity, soundness and completeness

*Property 1 (***Monotonicity [23]):** Given *symbolic* LSCs M = ($pre, full$) and M' = (pre', $full'$). If $full$ is a prefix of $full'$, every positive witness of M' is a positive witness of M. Hence, $sup(M') \leq sup(M)$.

The monotonicity property is used to prune the search space in our mining algorithms: after ascertaining that the support of a symbolic chart A is less than the minimum support threshold, one prune the search space of all charts having A as a prefix.

*Definition 1 (***Stat. Sound. & Completeness):** Given input dataset and thresholds, a mining algorithm is (1) *statistically sound* iff every mined result obeys the thresholds, and (2) *statistically complete* iff every potential result that obeys the thresholds is mined.

Following data mining algorithms in general, and frequent pattern mining algorithms in particular (see, [11]), our goal is to achieve statistical soundness and completeness, as described in Defn. 1. Our mining algorithms are sound and complete modulo the given traces and user-defined thresholds (see [26]).

IV. Towards Succinctness

To mine for succinct specifications, we introduce three concepts: equivalence class of LSCs and symbolic LSCs, isomorphic embeddings of statistically non-redundant LSCs, and summarization via the extraction of delta-discriminative LSCs.

A. Symbolic LSCs as equivalence classes

The first succinctness feature builds on the mapping from concrete to symbolic LSCs. A symbolic LSC corresponds to a set of concrete LSCs having the same structure, lifelines and methods among the lifelines. The only differences among members of the equivalence class are the actual objects that are bound to the lifelines.

An illustration of concrete and symbolic full-charts is shown in Fig. 4. The one on the left is the symbolic LSC and the others are some examples of corresponding concrete LSCs. This is the full-chart of the LSC shown in Fig. 3. We formalize concrete and symbolic charts and the mapping between them via the definition of *binding preserving abstraction* (BPA).

A concrete (symbolic) chart is composed of a list of concrete (symbolic) events $\langle e_1, e_2, \ldots, e_n \rangle$, a set of concrete (symbolic) lifelines $\{l_1, l_2, \ldots, l_k\}$ (for simplicity, we draw the lifelines ordered from left to right although the order of lifelines has no semantic meaning), and a binding function *bdg* mapping each event identified by its position in the chart to a pair of lifelines (we omit obvious syntactic well-formedness rules, e.g., that *bdg* binds an object to a lifeline only if the two are of the same class, etc.). More formally:

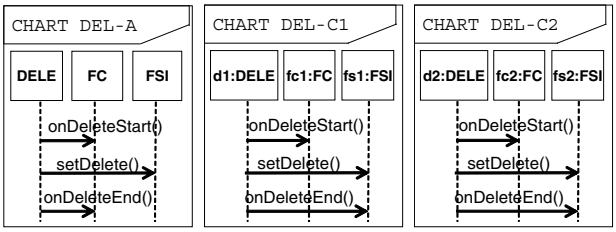

Fig. 4. **Charts: Symbolic and Concrete**

*Definition 2 (***Concrete (Symbolic) Chart):** A concrete (symbolic) chart C is a triplet $\langle E, L, bdg \rangle$ where E is a list of concrete (symbolic) events, L is a set of concrete (symbolic) lifelines, and *bdg* is a binding map $bdg : I \rightarrow L \times L$, where I = $\{i | 1 \leq$ i$\leq$ |E|$\}$ (i.e., the set of indices of events in E). Notation-wise, we refer to the pair of lifelines corresponding to the i-th element of E as $bdg[i]$.

Given a list of concrete events, a *single* set of concrete lifelines is determined by the names of concrete objects involved in the events. Thus, the binding map *bdg* of a concrete chart containing a list of concrete events (and a set of concrete lifelines, each corresponding to a unique concrete object) is trivial. While a list of concrete events uniquely determines a set of lifelines (up to lifelines order, which has no meaning), a trivial binding function, and thus, a concrete chart, this is *not* the case for symbolic events. Rather, given a list of symbolic events, one may possibly define more than one (non-isomorphic) sets of symbolic lifelines with corresponding bindings. This is so because a class corresponding to a symbolic caller or callee does not uniquely identify a symbolic lifeline in the set, as the chart may include a number of symbolic lifelines corresponding to (different objects of) the same class.

As an example, the binding of a concrete event e = ($oid1$, $oid2, m()$) in a concrete chart C, may be represented by a pair $\langle l_i, l_j \rangle$ corresponding to the ith and jth lifelines of C. $bdg(e,C)$ is the mapping returning the binding of a concrete event e from a concrete chart C. For example, in Fig. 5, $bdg((12:A, 15:A, m1())) = \langle 1, 2 \rangle$.

Therefore, to relate concrete and symbolic charts, we propose the notion of *binding preserving abstraction* (BPA), essentially an isomorphic mapping between the two.

*Definition 3 (***Binding Preserving Abstraction** *(BPA)):* Consider a concrete chart $CC = \langle \langle e_1, \ldots, e_n \rangle, \{l_1, \ldots, l_k\}, bdg \rangle$ and a symbolic chart $AC = \langle \langle E_1, \ldots, E_n \rangle, \{L_1, \ldots, L_k\}, BDG \rangle$. AC is a binding preserving abstraction of CC iff there exists a one-to-one mapping *abs* from the lifelines of CC to the lifelines of AC s.t. $\forall_{1 \leq i,j \leq k}$ and $\forall_{1 \leq v \leq n}$, $bdg(v) = \langle l_i, l_j \rangle$ iff $proj(e_v)$=E_v and $BDG(v) = \langle abs(l_i), abs(l_j) \rangle$.

We denote the binding preserving abstraction of CC by

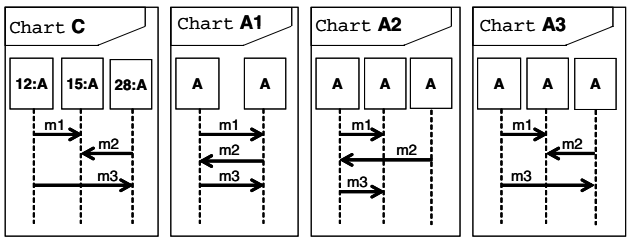

Fig. 5. **A Concrete Chart C and its binding preserving abstraction (BPA) A3. Other charts (A1 & A2) are not BPA of C.**

abs(*CC*). *abs*(*CC*) represents the symbolic chart that is 'isomorphic' to *CC*.

To illustrate the concept of binding preserving abstraction consider Fig. 5. In the figure there are a concrete chart (extreme left, each lifeline corresponds to a separate object instance of class *A* identifiable via the object hash code) and several symbolic charts. Two of the symbolic charts are not isomorphic to the concrete chart, and thus do not consist of a binding preserving abstraction.

The above concept is important as there can be more than one symbolic lifeline corresponding to a particular class *C* in a symbolic LSC *L*. When this is the case, more than one object instance of the class *C* participates in *L*.

We use the binding preserving abstraction *abs* to define an equivalence relation over concrete LSCs:

*Definition 4 (***Equivalence Relation):** Consider two concrete LSCs l_1 = L(pre_1,$full_1$) and l_2= L(pre_2,$full_2$). $l_1 \equiv_{abs} l_2$ iff abs(pre_1)=abs(pre_2), and abs($full_1$)=abs($full_2$).

It is easy to see that $\equiv_{abs}$ is indeed reflexive, symmetric, and transitive, and thus partitions any set of concrete LSCs into equivalence classes. We use the symbolic LSC corresponding to all concrete LSCs in a class as a representative of the class. To extract a succinct set of LSCs from an input trace database, we look for representative symbolic LSCs.

B. Isomorphic embeddings and redundancy

Following the concept of BPA, two charts are said to be isomorphic if the series of events are the same, and there is a one-to-one mapping between their lifelines that preserves the bindings. The formal definition is in Defn. 5. It is easy to see that a concrete chart having a chart C as a BPA will also have any isomorphic chart of C as a BPA.

*Definition 5 (***Isomorphic Charts):** Two charts f1 = $\langle E1, L1,\ bdg1\rangle$ and f2 = $\langle E2, L2,\ bdg2\rangle$ are isomorphic if $\exists$ a one-to-one mapping map from $L1$ to $L2$ such that $\forall i, bdg1[i] = (k_x, k_y)$ iff $E1[i] = E2[i] \wedge bdg2[i] = (map(k_x), map(k_y))$.

To illustrate isomorphic embeddings, consider the example chart in Fig. 6(left). This chart is an isomorphic embedding (i.e., it could be embedded isomorphically) of the chart shown in Fig. 4(left). On the other hand, the chart in Fig. 6(right) has no isomorphic embedding in the chart shown in Fig. 4(left). We formally define isomorphic embeddings below.

*Definition 6 (***Isomorphic Embeddings):** Consider charts C_A=$\langle\langle a_1,\ldots,a_n\rangle, \{l_{a_1},\ldots,\ l_{a_l}\}, bdg_a\rangle$ and C_B = $\langle\langle b_1,\ \ldots, b_m\rangle, \{l_{b_1},\ldots,l_{b_k}\}, bdg_b\rangle$. C_A has an isomorphic

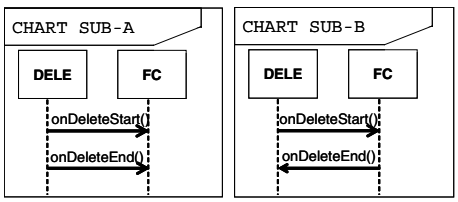

Fig. 6. **Isomorphic & Non-Isomorphic Charts**

embeddings within C_B iff there exists a mapping *iso* from the lifelines of C_A to the lifelines of C_B s.t. $\forall_{1\le v\le n},\ bdg_a(a_v) = \langle l_i, l_j\rangle$ iff $\exists_{1\le w\le m}.\ a_v = b_w \wedge bdg_b(b_w) = \langle iso(l_i), iso(l_j)\rangle$. We refer to C_B as a super-chart of C_A, denoted $C_B \sqsupseteq C_A$.

The above definition of isomorphic embeddings is used to define statistically redundant charts:

*Definition 7 (***Statistically Redundant):** Consider two LSCs l_1 = L(pre_1,$full_1$) and l_2 = L(pre_2,$full_2$). The first LSC l_1 is rendered redundant by the second LSC l_2 iff sup(l_1)= sup(l_2), conf(l_1)=conf(l_2) and $full_2 \sqsupseteq full_1$.

In our mining, we would like to eliminate statistically redundant charts. A chart is considered redundant if a super-chart of it, with the same statistics, appears in the mined set of charts. Note that a significant LSC is likely to have many significant sub-LSCs having the same support and confidence. By eliminating statistically redundant charts, many uninteresting charts are removed.[1]

C. Delta-discriminative summarization

Finally, the third summarization feature concerns the information gained by adding each LSC to a selected list of mined LSCs, with regard to other LSCs that were already selected to the list.

We start by sorting the mined LSCs in an ascending order based on number of lifelines, followed by total length (i.e., number of events in the LSC), followed by the inverse length of the pre-chart, followed by their confidence and their support. The sorted list of LSCs is then summarized by removing those LSCs which do not contribute enough additional information – in terms of new event signatures – relative to the ones ranked above them. As a result, similar LSCs with minor variations are removed from the final set of mined LSCs. The resulting succinct set is presented to the user.

*Definition 8 (***InfoGain(l_1,l_2)):** Consider two LSCs l_1 = L(pre_1,$full_1$) and l_2 = L(pre_2, $full_2$). Let $MSet_1$ and $MSet_2$ be the set of signatures contained in $full_1$ and $full_2$ respectively. The information gain of l_1 with respect to l_2 is defined as the ratio between $|MSet_1 \setminus MSet_2|$ and $|MSet_1|$.

We use the above to define delta-discriminative LSC.

*Definition 9 (***Delta-Discriminative):** Consider an ordered set of LSCs $ORD = \langle l_1,\ldots,l_n\rangle$. Given a minimum delta-discrimination threshold δ, an LSC $l_j \in ORD$ is delta discriminative iff $\nexists\, i < j$. InfoGain(l_i, l_j) $< \delta$. An ordered set of LSCs is delta-discriminative if all its LSCs are delta-discriminative.

[1]Note though, that statistical redundancy does not imply logical redundancy. In LSC semantics, a sub-chart and a super-chart are incomparable; one does not logically imply the other.

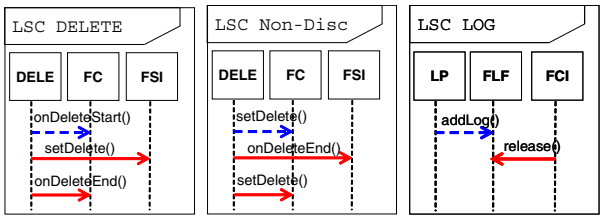

Fig. 7. **Non-Discriminative & Discriminative Charts**

Thus, to produce a succinct set of LSCs that covers many different method signatures, only a set of delta-discriminative LSCs are selected.

As an example, consider the LSCs shown in Fig. 7 ordered from left to right. With delta-discriminative threshold set at 10%, the second LSC would not be selected while the third LSC would. For every selected representative LSC, other similar LSCs with minor variations are removed.

D. Additional considerations: no repetitions and connectedness

In addition to the above three features, we provide filters that could optionally be applied to restrict the format of mined LSCs to ones commonly used in the literature. We find that often methods are not repeated in an LSC, hence we introduce a filter to remove LSCs with repeated methods. An example of such LSC is shown in Fig. 7(middle) where method setDelete() is repeated twice. In addition, we remove LSCs that are not monotonically connected. An LSC L is monotonically connected if for every prefix of L, each lifeline in the prefix is connected, either directly or indirectly, to another lifeline. We denote that an LSC L is monotonically connected by $MCONN(L)$. By default, these additional filters are employed.

Problem Statement. Given support, confidence, and delta discriminativeness thresholds, mine all *symbolic* LSCs that are *non-redundant* and *delta discriminative*. We refer to the this set of LSCs as the set of *succinct* LSCs.

V. Mining Framework

This section describes our technique to mine a succinct set of LSCs from a set of traces, starting with the framework overview and continuing with the algorithm in detail.

A. Mining Framework

The mining framework starts with a program and ends with a succinct set of mined LSCs. The program under investigation is instrumented with 'print' statements at the entries of selected method calls; different instrumentation techniques could be used ranging from binary instrumentation, to byte code injection to aspect oriented programming. When executed, the instrumented program produces a set of traces. If a set of test cases is present, these test cases could be run to produce a set of traces. Otherwise, typical user interaction with the user interface component of the system could be performed and a corresponding set of traces could be collected.

The set of traces is fed to the miner (Part 2,3 and 4). The miner finds every LSC $L(pre, full)$ that satisfies the following criteria: (1) The full chart $full$ is observed at least min_sup number of times in the set of traces; (2) when the pre-chart

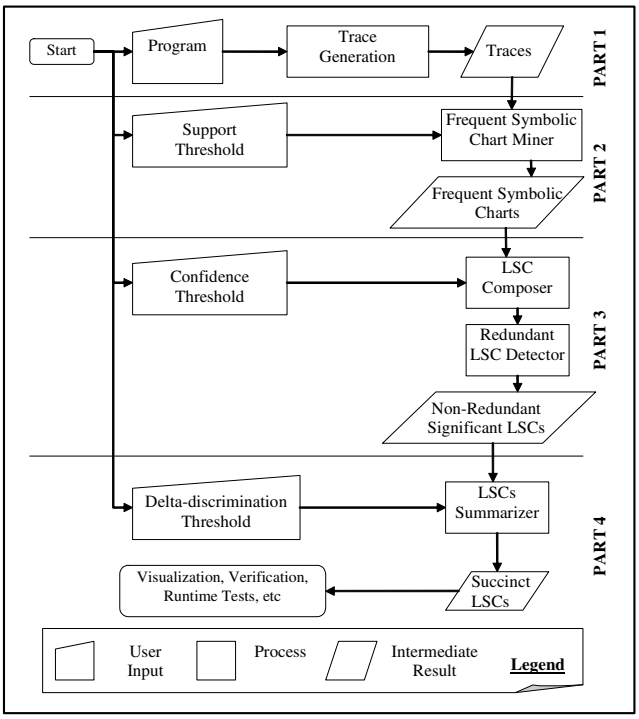

Fig. 8. **Mining Framework**

pre occurs, the main-chart would eventually occur with at least min_conf likelihood in the input traces; (3) the set is non redundant; and (4) all LSCs are delta-discriminative with regard to higher ranked mined LSCs.

The mined LSCs could be visualized to help program understanding, fed to runtime monitoring tools [26], [29], used for software verification, etc.

B. Mining Algorithm

We start by describing a basic operation used during the mining process. It then follows with the algorithm to mine frequent symbolic charts. These charts are later composed to form significant LSCs. Redundant LSCs are removed on-the-fly during the composition process. As a final process, the mined significant non-redundant charts are further ranked and summarized. The complexity analysis of the various procedures in the algorithm is available in [1].

Basic Operation. We introduce a basic operation termed as *possible bindings* that computes the possible extensions of a symbolic chart given an additional symbolic event to append to the chart. We define it formally in Definition 10.

*Definition 10 (***Possible Bindings):** Given a symbolic chart AC and a symbolic event AE, concatenating AE to AC results in a *set* of (non-isomorphic) possible symbolic charts. Each of the charts in the set corresponds to a different valid pair of symbolic lifelines assigned as the bindings of AE. We denote the set of possible resultant charts from AC and AE by $PBDG(AC,AE)$.

To illustrate the set $PBDG(\cdot,\cdot)$, consider the left-most chart Abs in Fig. 9. Computing $PBDG(Abs, (B,A,m2()))$ returns four possible charts: $AbsExt$-1,2,3 and 4 in Fig. 9.

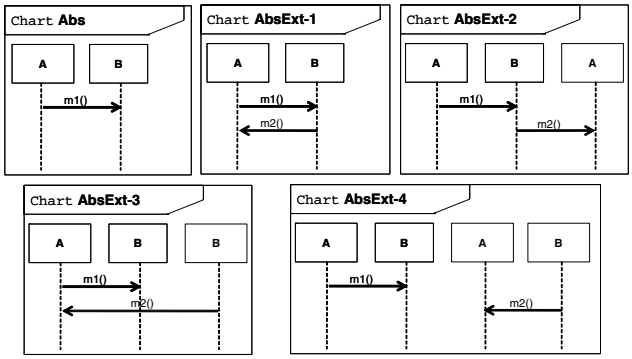

Fig. 9. PBDG Examples

Procedure MineFreqCharts
Inputs:
TDB : Input Trace Database
min_sup: Min. Sup. Thresh.
Output:
A set of frequent charts
Method:
1: Let E = Single-event charts satisfying *min_sup*
2: For each f_E in E
3: Call MineWthPrefix(*min_sup*,E,f_E)

Procedure MineWthPrefix
Inputs:
min_sup: Min. Sup. Thresh.
E: Frequent single events
CurC: Current chart considered
Method:
4: Output *CurC*
5: For each f_E in E
6: Let *nxCSet* = *PBDG*(C,f_E)
7: For each *nxC* in *nxCSet*
8: If ($|TDB_{nxC}| \geq min\_sup \wedge MCONN(nxC)$)
9: Call MineWthPrefix(*min_sup*,E,*nxC*)

Fig. 10. **Mine Frequent Charts Procedure**

Mining Frequent Symbolic Charts. Our mining algorithm works in a depth first manner. First, chart of length 1 is considered. Consecutively, we incrementally increase the length of the chart by appending events to the previously considered charts. Following Property 1, when a chart under consideration is infrequent, all its extensions (i.e., all charts having this chart as a prefix) would be infrequent too. Hence, we safely stop growing the chart further when we find that a chart is infrequent. With this we prune a large search space containing infrequent charts. Furthermore, all additional filters, e.g., monotonic connectedness, etc., are applied at this phase when LSCs are composed from frequent charts. The algorithm to mine all symbolic charts is shown in Fig. 10.

Extraction of Non-Redundant Significant LSCs. Given a set of symbolic charts, significant LSCs are formed by composing symbolic charts. Two charts *pre* and $full$, where *pre* is an isomorphic prefix of full, could be composed to form LSC $L(pre, full)$ with support $sup(full)$ and confidence $sup(full)/sup(pre)$. We are interested only in LSCs whose confidence is above the minimum confidence threshold. During composition we detect redundant LSCs on the fly. As defined in Defn. 7, an LSC L is redundant if there exists another LSC L' where there is an isomorphic embedding of L in L' and both LSCs have the same statistics (i.e., equal support and confidence).

Procedure ComposeNRLSCs
Inputs:
CHT : All frequent charts
min_conf: Min. Conf. Thresh.
Output:
A set of non-redundant significant LSCs
Method:
1: Initialize $BKTS$ as a hash table
2: For each $full$ in CHT
3: For each isomorphic prefix *pre* of $full$
4: Create LSC $L(pre,full)$
5: If ($conf(L) \geq min\_conf$)
6: Put L to $BKTS$ based on its support and conf.
7: Let *CandNR* = bucket's candidate non-red. LSCs
8: Update *CandNR* accordingly – see text
9: For each bucket in BKTS
10: Output all non-redundant LSCs in the bucket

Fig. 11. **Compose Non-Redundant Significant LSCs Procedure**

A straightforward approach to extract non-redundant LSCs would be to compare each LSC with all other LSCs. However, this would be expensive – $O(n^2)$. Since n could be large, we would like to have a more efficient approach. First, since we would like to check only the LSCs having the same support and confidence, we hash the LSCs based on their support and confidence values. Only LSCs in the same bucket would need to be compared with one another.

To further speed up the process, we check for redundant LSCs on-the-fly when new LSCs are formed and added to the bucket. A temporary data structure is used to hold candidate non-redundant LSCs. Whenever an LSC in the bucket is considered, this LSC is checked only against all candidate LSCs found so far. If there exists a candidate LSC that is a super-chart of the new LSC, this LSC would not be added as a candidate. Otherwise, it is added. During the process, candidate LSCs that are found to be embedded in newly formed LSCs are removed from the list of candidates. At the end, the candidate LSCs in each bucket would be the set of non-redundant LSCs with that particular statistics (i.e., support and confidence values).

Checking for redundant LSCs builds upon the checking of isomorphic embeddings. We move the description of the checking of isomorphic embeddings to [1].

Ranking and Summarization. To perform the summarization, we first sort the LSCs in an ascending order based on number of lifelines, total length (i.e., number of messages), the inverse length of the premise, their confidence and their support. The sorted list is then summarized by removing those LSCs which do not give enough extra information (in terms of added messages) relative to the ones ranked above them. The delta-discrimination threshold is defined by the user; only those LSCs that have at least δ percent difference from LSCs ranked above them would be left in the summary. More details are available in [1].

Complexity. We divide the complexity analysis into three parts: the computation of frequent symbolic charts, the compu-

tation of non-redundant symbolic LSCs, and the computation of succinct LSCs. The complexity of computing frequent symbolic charts is linear in the number of frequent symbolic charts found. The complexity of computing non-redundant symbolic LSCs is linear to number of frequent symbolic charts and the largest bucket size used. The complexity of computing succinct LSCs is $O(nLog(n))$ where n is the number of symbolic non-redundant LSCs.

Output Guarantee. The above algorithm guarantees that all specifications mined are succinct (i.e., sound), and all succinct specifications are mined (i.e., complete). The definition and discussion on statistical soundness and completeness is given in Section III-D.

Theorem 1: Our algorithm produces a statistically sound and complete set of succinct LSCs (i.e., it is significant, symbolic, non-redundant, delta-discriminative and obeys the set of filters specified by users).

Proof sketch Soundness is guaranteed as we check each LSC output to see if it is succinct.

Completeness is guaranteed due to the following. Our framework considers the search space of all possible LSCs in a depth first search manner. To make mining feasible, a pruning strategy based on Property 1 is used to cut the search space of non-frequent charts. Additional search space might be pruned based on the filters described in Section IV-D. Since only search spaces consisting of non-succinct LSCs (i.e., those not satisfying minimum support or are deemed not interesting based on the filteres) are pruned, and we exhaustively traversed all search space that are not pruned, our algorithm produces a complete set of succinct LSCs. □

Given a specification format, a set of traces and some criteria (i.e., support, confidence, delta discrimination, etc), we would mine all specifications from the traces obeying the format and criteria. These properties are commonly obeyed by data mining tools [11] and invariant generation tools like Daikon [9]. Admittedly, just like other dynamic analysis tools, the quality of the mined specifications would vary depending on the traces. This is common to dynamic analysis tasks and to data mining tools: results can only be as good as the quality of the input data.

VI. Experimental Results

We have implemented the ideas presented in this paper and evaluated them on two case study applications. All experiments were performed on a Intel Core 2 Duo 2.40GHz Tablet PC with 3.24 GB of RAM running Windows XP Professional. Algorithms were written using Visual C#.Net running under .Net Framework 2.0 with generics compiled using Visual Studio.Net 2005.

We first analyzed crossFTP server [5], a commercial open source FTP server built on top of Apache FTP server. The server consists of 15 packages containing 1148 methods in 165 classes spanning 18841 LOC. We used AspectJ to generate traces suitable for our needs, running the server with usage scenarios involving file transfers, deletions, renames, closing/ opening connections, starting/ closing the server etc. We collected 54 traces with a total length of 1876 events.

App.	MSup	Time(s)	|Syb.|	|NR|	|Suc.|
Jeti	10	527	323,269	50	16
Jeti	15	232	127,903	24	9
Jeti	20	1	20	7	6
CrossFTP	20	1,640	118,012	12	11
CrossFTP	30	1,635	115,234	5	4
CrossFTP	40	1,636	115,234	5	4

TABLE II
EXPERIMENT RESULT BY VARYING TRACE SETS AND MINIMUM SUPPORT THRESHOLDS WITH MIN_CONF=100%, AND MIN_DISC=10%. THE COLUMNS CORRESPOND TO THE APPLICATION WHERE THE TRACES COME FROM (APP.), THE MINIMUM SUPPORT THRESHOLD (MSUP), THE NUMBER OF SYMBOLIC LSCS (|SYB.|), THE NUMBER OF NON-REDUNDANT SYMBOLIC LSCS (|NR|), AND THE NUMBER OF SUCCINCT LSCS (I.E., SYMBOLIC, NON-REDUNDANT, AND DELTA-DISCRIMINATIVE LSCS) MINED (|SUC.|), RESPECTIVELY.

We also analyzed Jeti [2], a popular full featured open source instant messaging application. Jeti has an open plug-in architecture. It supports many features including file transfer, group chat, buddy lists, presence indicators, etc. Its core contains 49K LOC consisting of about 3400 methods, 511 classes in 62 packages. We collected 5 traces with a total length of 1797 events.

Succinctness Experiments. We vary the support values for the two sets of traces and run the mining. Table II contains information on the time needed to mine the scenarios for the various support thresholds, and the number of symbolic LSCs, non-redundant symbolic LSCs, and succinct LSCs (i.e., symbolic, non-redundant, and delta-discriminative LSCs) mined.

From Table II, running on traces from Jeti at the minimum support, confidence and delta-discrimination threshold set at 10, 100% and 10% respectively, the algorithm completed within 10 minutes and a total of 16 succinct scenarios were mined. Note that mining symbolic LSCs without the redundancy filter and summarization produces 323,269 LSCs. That is, the algorithm reduced the number of LSCs mined by a factor greater than 10,000; many concrete LSCs could be merged into one symbolic LSC; many redundant and non-delta discriminative LSCs were removed.

In addition, for traces of CrossFTP at the minimum support, confidence and delta-discrimination threshold set at 20, 100% and 10% respectively, the algorithm completed within 35 minutes and a total of 11 succinct scenarios were mined. Note that mining symbolic LSCs without the redundancy filter and summarization produced 118,012 LSCs. That is, the algorithm reduced the number of LSCs mined by a factor greater than 10,000; again many concrete LSCs could be merged into one symbolic LSC; many redundant and non-delta discriminative LSCs were removed.

For Jeti traces, at higher support threshold (i.e., 20) the number of the mined LSCs is smaller. This affects also the number of redundancies found. Still the eventual number of succinct LSCs mined is smaller (6 instead of 9 or 16), showing that indeed some information was lost due to the use of the initial higher support threshold. For CrossFTP, even

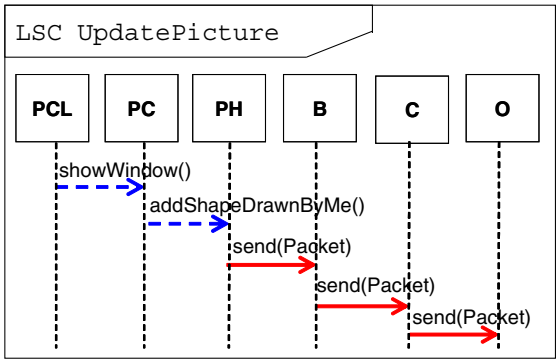

Fig. 12. **[Jeti] Mined LSC: Update Picture**

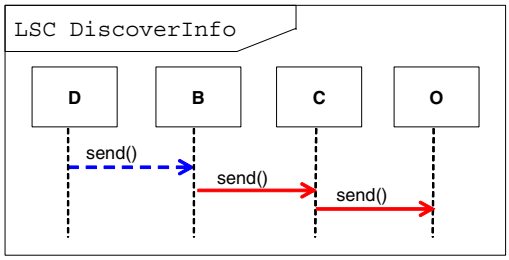

Fig. 13. **[Jeti] Mined LSC: Discover Information**

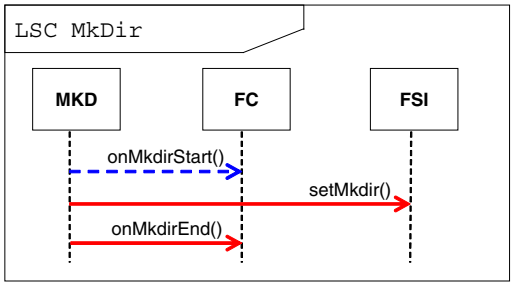

Fig. 14. **[CrossFTP] Mined LSC: Make Directory**

with higher support thresholds the number of redundant LSCs remains high as there are large mined LSCs with high support.

Some Mined Scenarios. We show some mined scenarios from Jeti and CrossFTP. Figs. 12 & 13 show two scenarios related to the drawing and internal information discovery functionalities of Jeti. Fig 12 shows that *whenever PictureChangeListener (PCL) (i.e., an object of type PCL) calls the showWindow() method of PictureChat (PC) and PC calls addShapeDrawnByMe() of PictureHistory (PH), it is the case that PH calls calls the send(Packet) method of Backend (B) which is then relayed by calling send(Packet) methods of Connect (C) and Output (C)*. Fig 13 shows that *whenever Discovery (D) calls send(Packet) method of Backend (B), the packet is relayed via the send(Packet) methods of Connect (C) and Output (O)*.

Fig. 14 shows an LSC mined from CrossFTP. It describes the scenario of creating a new directory on the server, involving an MKDIR command, an FC, and a FSI objects. It is interesting to note that two other mined scenarios, for moving and renaming files (not shown here), were also mined. The three scenarios share a very similar structure, differing only in the identity of the first lifeline and the names of the methods involved. As future work, it may be interesting to develop additional abstraction operators that will allow us to automatically recognize such similar structures and thus report them in a more succinct way.

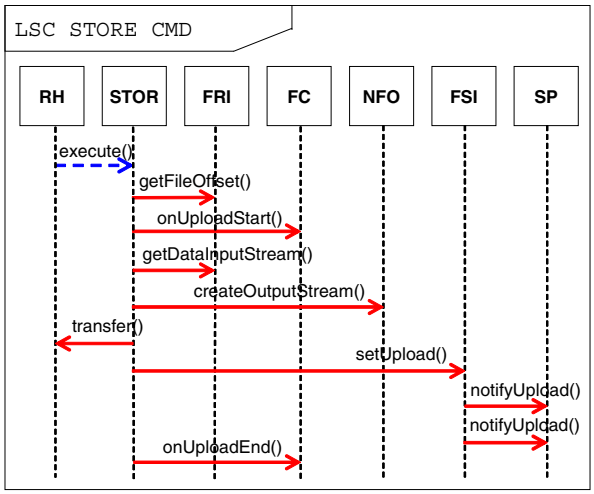

Fig. 15. **[CrossFTP] Mined LSCs: Upload**

Figs. 15 tells the 'story' of uploading a file to the server: *whenever a RequestHandler (RH) object calls the execute() method of a STOR command object, the latter eventually calls the getFileOffset() method of a FtpRequestImpl (FRI) object, the onUploadStart() method of a FtpletContainer (FC) object, the getDataInputStream() method of the FRI object, and the createOutputStream() of a NativeFileObject (NFO) object. Later the STOR command object calls the transfer() method of the RequestHandler (RH) object and the setUpload() method of a an FtpStatisticsImpl (FSI) object. Finally, the FSI object makes two calls to the notifyUpload() method of a StatisticsPanel (SP) object (each with a different signature, abstracted away in the diagram), and the scenario ends with the STOR command object calling the onUploadEnd() of the FC object*. Note how this mined LSC gives a comprehensive inter-object picture of the way upload requests are handled by the server. Also note the use of symbolic lifelines (notice the indefinite 'a' and definite 'the' in the description). Finally, recall that this scenario was extracted from a number of traces, where many events related to other features interleaved with the events in the scenario, including the simultaneous uploading of a number of files. Thus, abstraction from concrete to symbolic LSC had to be correctly employed.

Additional details of the case studies are available in [1].

VII. Related Work

Many studies propose variants of dynamic analysis based specification mining (e.g., [9] mine boolean expressions describing likely invariants on values of program variables, [4], [15], [31], [33] mine temporal orderings of method calls/network packets as finite state machines, [8], [32] mine frequent patterns of behavior, [7] mine implied message sequence charts). Different from the above mentioned studies, we mine a set of LSCs from traces of program executions. We believe sequence diagrams in general and LSCs in particular, are suitable for the specification of inter-object behavior, as

they make the different role of each participating object and the communications between the different objects explicit. LSCs also express modalities in the relationship between the pre- and the main- chart. Our work is not aimed at discovering the complete behavior or APIs of certain components, but, rather, to capture the way components cooperate to implement certain system features. Indeed, inter-object scenarios are popular means to specify requirements (see, e.g., [12]).

The challenge of succinctness in specification mining is not limited to the scenario-based approach. Some work present mining of rules and patterns (see, e.g., [8], [32], [20], [27]). At times, the number of rules and patterns could be very large. This makes it hard for users to investigate mined specifications as there are many reported rules and patterns which are very similar to one another. In Daikon [9], the number of reported invariants could also be too large. In order to address this issue, according to Daikon's documentation, Daikon may be configured to remove logically redundant invariants by using a theorem prover. In mining finite-state machines (e.g., [4], [28]), at times the size of the automaton could be too large to be useful, with too many connections between nodes. This could happen especially if various unrelated sub-specifications are interleaved together.

To address the large number of patterns/rules mined, studies in [8], [17], [20], [21], [27] remove redundant patterns/rules in the process. In this work, to eliminate redundancy we consider not only the sequences of method calls but also lifelines and isomorphic relationships among charts which make the task more challenging. In a workshop paper, our preliminary work introduces the concept of mining symbolic LSCs [23], however it is not implemented and there is no experimental study. In this work, we extend the concept further by introducing redundancy among symbolic LSCs and delta-discriminative summarization. We also provide an implementation which we use to experiment on two case studies.

Hamou-Lhadj and Lethbridge propose an approach to summarize traces by removing uninteresting events [10]. Different from their work, we perform summarization on the mined LSCs rather than on the traces. The two approaches are orthogonal; we could use the approach in [10] to help in selecting the important events that we should trace.

In this work, we only consider statistical redundancy. In the future, it is interesting to combine our approach with work on logical redundancy, e.g., [3]. One could also construct a subsumption relation based on software hierarchies, e.g., object composition hierarchy, package hierarchy, etc, and mine for representative LSCs that subsume others. In effect, this could be viewed as an integration of this work with our previous work in [24], which proposes an approach to zoom-in and -out mined specifications. In [25], we introduce mining scenario-based specifications with value-based invariants involving method parameters. In this work, we focus on LSCs without value-based invariants. In the future, we plan to integrate these two approaches together.

VIII. Conclusion and Future Work

In this paper we addressed the succinctness challenge of specification mining in the context of the scenario-based approach, mining a succinct set of significant LSCs from a set of program execution traces. We formulated and evaluated a definition of an equivalence relation over LSCs, a definition of a redundancy and inclusion relation based on isomorphic embeddings among LSCs, and a delta-discriminative measure based on an information gain metric on a sorted set of LSCs, all used on top of the statistical metrics of support and confidence. Our case studies showed the utility of our work in reducing the number of mined LSCs. Future work includes potential generalization and application of the methods we presented in this paper to other specification mining approaches.

References

[1] *www.mysmu.edu/faculty/davidlo/succinct/succinct.html*.
[2] Jeti. Version 0.7.6 (Oct. 2006). http://jeti.sourceforge.net/.
[3] P. Abdulla, Y.-F. Chen, L. Clemente, L. Holk, C.-D. Hong, R. Mayr, and T. Vojnar. Simulation subsumption in Ramsey-based Buchi automata universality and inclusion testing. In *CAV*, 2010.
[4] G. Ammons, R. Bodik, and J. R. Larus. Mining Specification. In *POPL*, 2002.
[5] CrossFTPServer. sourceforge.net/projects/crossftpserver/.
[6] W. Damm and D. Harel. LSCs: Breathing Life into Message Sequence Charts. *J. on Formal Methods in System Design*, 19(1):45–80, 2001.
[7] F. C. de Sousa, N. C. Mendonça, S. Uchitel, and J. Kramer. Detecting implied scenarios from execution traces. In *WCRE*, 2007.
[8] M. El-Ramly, E. Stroulia, and P. Sorenson. Interaction-pattern mining: Extracting usage scenarios from run-time behavior traces. In *KDD*, 2002.
[9] M. Ernst, J. Cockrell, W. Griswold, and D. Notkin. Dynamically discovering likely program invariants to support program evolution. *IEEE TSE*, 27(2):99–123, February 2001.
[10] A. Hamou-Lhadj and T. Lethbridge. Summarizing the content of large traces to facilitate the understanding of the behaviour of a software system. In *ICPC*, 2006.
[11] J. Han and M. Kamber. *Data Mining Concepts and Techniques*. Morgan Kaufmann, 2006.
[12] D. Harel. From play-in scenarios to code: An achievable dream. *IEEE Computer*, 34(1):53–60, 2001.
[13] D. Harel, H. Kugler, and A. Pnueli. *Synthesis Revisited: Generating Statechart Models from Scenario-Based Requirements*, pages 309–324. Springer, 2005.
[14] D. Harel and S. Maoz. Assert and Negate Revisited: Modal Semantics for UML Sequence Diagrams. *Software and Systems Modeling*, 7(2):237–252, 2008.
[15] H. Hungar and B. Steffen. Behavior-based model construction. *STTT*, 6(1), 2004.
[16] M. Huth and M. Ryan. *Logic in Computer Science*. Cambridge, 2004.
[17] H. Kagdi, M. Collard, and J. Maletic. An approach to mining call-usage patterns with syntactic context. In *ASE*, 2007.
[18] H. Kugler, D. Harel, A. Pnueli, Y. Lu, and Y. Bontemps. Temporal Logic for Scenario-Based Specifications. In *TACAS*, 2005.
[19] M. Lettrari and J. Klose. Scenario-Based Monitoring and Testing of Real-Time UML Models. In *UML*, 2001.
[20] D. Lo, S.-C. Khoo, and C. Liu. Efficient mining of iterative patterns for software specification discovery. *KDD*, 2007.
[21] D. Lo, S.-C. Khoo, and C. Liu. Mining temporal rules for software maintenance. *J. of Software Main. and Evolution*, 2008.
[22] D. Lo and S. Maoz. Mining Scenario-Based Triggers and Effects. In *ASE*, 2008.
[23] D. Lo and S. Maoz. Mining Symbolic Scenario-Based Specifications. In *PASTE*, 2008.
[24] D. Lo and S. Maoz. Mining hierarchical scenario-based specifications. In *ASE*, 2009.
[25] D. Lo and S. Maoz. Scenario-based and value-based specification mining: Better together. In *ASE*, 2010.
[26] D. Lo, S. Maoz, and S.-C. Khoo. Mining Modal Scenario-Based Specifications from Execution Traces of Reactive Systems. In *ASE*, 2007.
[27] D. Lo, G. Ramalingam, V.-P. Ranganath, and K. Vaswani. Mining Quantified Temporal Rules: Formalism, Algorithms, and Evaluation. In *WCRE*, 2009.
[28] D. Lorenzoli, L. Mariani, and M. Pezzè. Automatic Generation of Software Behavioral Models. In *ICSE*, 2008.
[29] S. Maoz and D. Harel. From multi-modal scenarios to code: compiling LSCs into AspectJ. In *FSE*, 2006.
[30] R. Marelly, D. Harel, and H. Kugler. Multiple Instances and Symbolic Variables in Executable Sequence Charts. In *OOPSLA*, 2002.
[31] J. Quante and R. Koschke. Dynamic protocol recovery. In *WCRE*, 2007.
[32] H. Safyallah and K. Sartipi. Dynamic Analysis of Software Systems using Execution Pattern Mining. In *ICPC*, 2006.
[33] M. Shevertalov and S. Mancoridis. A reverse engineering tool for extracting protocols of networked applications. In *WCRE*, 2007.