Mining Scenario-Based Triggers and Effects

David Lo[1]
Singapore Management University
Email: davidlo@smu.edu.sg

Shahar Maoz
The Weizmann Institute of Science, Rehovot, Israel
Email: shahar.maoz@weizmann.ac.il

Abstract

We present and investigate the problem of mining scenario-based triggers and effects from execution traces, in the framework of Damm and Harel's live sequence charts (LSC); a visual, modal, scenario-based, inter-object language. Given a 'trigger scenario', we extract LSCs whose pre-chart is equivalent to the given trigger; dually, given an 'effect scenario', we extract LSCs whose main-chart is equivalent to the given effect. Our algorithms use data mining methods to provide significant sound and complete results modulo user-defined thresholds. Both the input trigger and effect scenarios, and the resulting candidate modal scenarios, are represented and visualized using a UML2-compliant variant of LSC. Thus, existing modeling tools can be used both to specify the input for the miner and to exploit its output. Experiments performed with several applications show promising results.

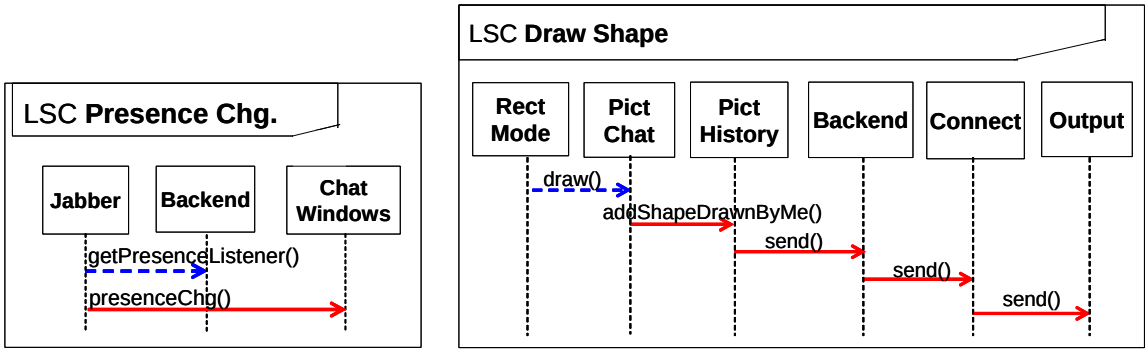
1. Introduction

Specification mining is a dynamic analysis process aimed at extracting candidate specification models of a program from its execution traces (see, e.g., [6], [30]), to aid, e.g., in testing, debugging, and program comprehension. Specifically, in previous work [24] we have introduced a framework, a method, and prototype implementation for mining of modal scenarios, in the form of a UML2-compliant variant of Damm and Harel's live sequence charts (LSC) [8], [12]. There, we used a data mining algorithm to mine a sound and complete set of statistically significant universal LSCs modulo given execution traces and user defined thresholds of support and confidence metrics. The popularity and intuitive nature of sequence diagrams as a specification language in general, together with the the additional unique features of LSC, motivated our choice of the target formalism of our mining approach. Moreover, the choice is supported by previous work on LSC (see, e.g., [15], [19], [26]), which can be practically used to visualize, analyze, manipulate, test, and verify the specifications we mine.

To introduce the problem of triggers and effects mining investigated in this paper consider the following two examples, taken from Jeti [4], an instant messaging application we experiment with. When a user starts a drawing on the shared white board, the `draw()` method of one of the shapes is called. A programmer investigating a bug in Jeti, 'why some shapes are not sent to the other party?', may be interested in finding the sequence of events which always occur in correct executions following this call. Moreover, when such a sequence is found, the programmer may be interested in transforming it into a runtime monitor, to track the relevant events in future executions. Conversely, a Jeti user's presence indicator changes whenever the user's online status changes (offline, busy, etc.). A programmer responsible for a change request, to conditionally disable this functionality in Jeti based on user preferences, may be interested in the sequences of method calls which, whenever occur, are followed by a call to the `presenceChg()` method of the `ChatWindow`. Moreover, when such a sequence is found, the programmer may be interested in considering it as a candidate property for formal verification of Jeti, before and after the changes will be made. Our work uses dynamic analysis of execution traces to provide the programmer with the means for investigating the answers to these questions and addressing similar property discovering tasks for debugging, evolution, testing, and verification.

Thus, in this paper we focus on a special case of scenario-based specification mining – mining of trigger and effect scenarios – which we find both more useful and more practical than the general problem. Given a 'trigger scenario', we extract significant LSCs whose pre-chart is equivalent to the given trigger; dually,

1. The work was done while the author was full time at School of Computing, National University of Singapore.



(a) (b)

Figure 1. Mined LSCs: (a) Presence Changed, (b) Draw Shape

given an 'effect scenario', we extract significant LSCs whose main-chart is equivalent to the given effect. Fig. 1 shows two LSCs, resulting from our experiments with Jeti, and answering the example questions given in the previous paragraph.

The problem of mining triggers and effects addressed here is very different than the general scenario-based specification mining problem considered in [24], where all significant LSCs meeting support and confidence thresholds are mined. In contrast, in this work, given a user-defined trigger or effect chart, only the corresponding effects or triggers are automatically computed. Instead of requiring an arbitrary support threshold, a normalized and more intuitive support threshold is used (see Sec. 3). Also, with scenario-based trigger and effect mining, the search space of possible LSCs is restricted – the longer the given effect or trigger the more restricted the search space becomes. This difference is evident in the performance of the mining algorithm, and thus allows us to handle significantly longer traces (see the experiments in Sec. 4). Finally, from a qualitative methodological perspective, the current work suggests a different, more focused and interactive specification mining process.

As inputs, the mining algorithm accepts a set of traces of inter-object events, a trigger or an effect, and a set of statistical significance thresholds. The algorithm models mining as a search space exploration process. Effective search space pruning strategy based on a monotonicity property of a support statistic is employed to efficiently extract triggers (or effects) from the input execution trace and the given effect (or trigger). Simply put, effects mining first finds all instances of the given trigger and then look forward for significant effects. Trigger mining is harder, as it is not a direct inverse of the problem of mining effect; looking 'backward' does not provide the required result (in future-time temporal expression) but rather a past-time temporal expression. Our solution is to first find significant triggers of length 1, and then grow them forward using the events in between the instances of the trigger and its given effect in the trace (see Sec. 3).

It is important to note that scenario-based specification mining is *not* aimed at finding a complete specification of the system under investigation. The scenario-based approach to modeling, in general, is not aimed at providing complete systems specifications. Rather, the strength of the scenario-based approach to modeling is that it allows the specifier to brake up the spec into 'pieces of behavior' or 'scenarios', each of which cuts across multiple objects. Moreover, trigger/effect mining is intrinsically partial, focusing on a specific behavior the user is interested in, such as a certain aspect of the program or a specific feature or bug; this focus on behaviors of interest is an important characteristics of our work.

Finally, our work on mining triggers and effects is an example for and part of a larger framework we are working on, presenting and examining what we may call *user guided specification mining*, an iterative interactive process, which allows the user to take advantage of prior knowledge and knowledge acquired from previous mining sessions in order to shape and direct the next mining task, focusing and narrowing down on the aspects of the specification that require examination for a specific purpose.

In this context, we consider the potential end-to-end visual characteristics of our work to be a major advantage. That is, both the input for the miner, trigger or effect sub scenarios, and the output of the miner, a set of LSCs, may be created and presented visually, within an industry standard modeling tool (in our case, IBM RSA [3]). We believe this will significantly contributes to the usability of our work and its accessibility to software engineers.

The paper is organized as follows. Basic concepts

and mining algorithms outlines are presented in the next two sections. Sec. 4 presents experimental results and evaluation. Sec. 5 discusses advanced features and considerations. Sec. 6 discusses related work and Sec. 7 concludes.

2. Concepts & Definitions

We briefly recall LSC, describe the semantics of triggers and effects, and define the metrics of support and confidence used in our work as the basis for evaluating candidate triggers and effects significance.

Live Sequence Charts We use a restricted subset of the LSC language. An LSC 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. Syntactically, instance lifelines are drawn as vertical lines, pre-chart (main-chart) events are colored in blue (red) and drawn using a dashed (solid) line. Semantically, 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 in between the events that are mentioned in the chart). For a thorough description of the language and its semantics see [8], [10]. A UML2-compliant variant of the language using the *modal* profile is defined in [12]. A translation of LSC into various Temporal Logics appears in [17]. The diagrams shown in our current work follow vertical ordering, that is, events are strictly ordered from top to bottom with no partial order.

An additional important feature of LSC is its semantics of *symbolic instances* [27]. Rather than referring to concrete objects, instance lifelines may be labeled with a name of a class and defined as symbolic, i.e., formally representing any object of the referenced class. This allows a designer to take advantage of object-oriented inheritance and create more expressive and succinct specifications.

Fig. 2 shows an example LSC taken from Jeti. The participants in this scenario are `UndoAction`, `PictureChat`, `PictureHistory`, `Backend`, `Connect`, `Output`, and `PictureChangeListener`. Roughly, this LSC specifies that *"whenever an object of type* `UndoAction` *calls an object of type* `PictureChat` `undo()` *method, eventually the history is updated (by calling* `undoMyAction()` *method of* `PictureHistory`*), messages are sent to the other party (the 3* `send()` *method calls), and* `PictureChat` *dialog is refreshed (the* `showWindow()` *method call)"*. We denote an LSC by $L(pre, main)$, where pre denotes the pre-chart and $main$ denotes the main chart.

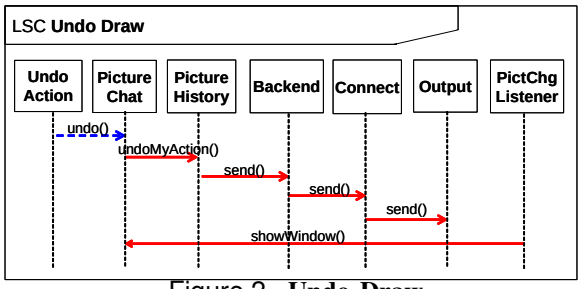


Figure 2. **Undo Draw**

Semantics of triggers and effects Roughly, as explained above, an LSC is composed of two charts, pre- and main-chart. LSC specifies a universal temporal rule: "Whenever the pre-chart is satisfied, eventually the rest of the LSC must be satisfied". Before formally defining triggers and effects, we precisely define the meaning of charts satisfaction.

We have two types of events: concrete and abstract. A *concrete event* is a triplet: caller object identifier, callee object identifier, and method signature. An object is uniquely identifiable usually by its hash key and the class it is instantiated from. The input trace is simply a series of concrete events. An *abstract event* is a triplet: caller class identifier, callee class identifier, and method signature. An abstract event groups several related concrete events together: it replaces object identity with a higher level of abstraction, in our case, the object's class.

A bit more formally, let Σ_{con} and Σ_{abs} be the set of concrete events and abstract events, respectively. Each concrete event in Σ_{con} corresponds to a unique abstract event in Σ_{abs}. Each abstract event in Σ_{abs} corresponds to a set of events from Σ_{con}. The map from a concrete event to its abstract event is a projection. Given a concrete event o, $proj(o)$ returns the abstract event of o. Events can be sequentially ordered and composed to form a chart.

*Definition 2.1 (***Concrete Chart):** A concrete chart CC corresponds to a series of concrete events $\langle o_1, \ldots, o_n \rangle$. Each event in turn is a triplet containing: caller object identifier, callee object identifier and method signature.

A set of concrete charts may map to a single abstract chart. We define a simple projection mapping from

concrete and abstract chart.

*Definition 2.2 (***Abstract Chart):** A concrete chart CC $\langle o_1, \ldots, o_n \rangle$ is mapped to an abstract chart AC $\langle a_1, \ldots, a_n \rangle$ iff $\forall_{1 \leq i \leq n}$. $proj(o_i) = a_i$. We define the mapping from concrete to abstract chart by extending the projection operator to charts, *i.e.*, $proj(CC) = AC$.

Two or more charts can be concatenated together. The concatenation operator is defined below.

*Definition 2.3 (***Concatenation):** Consider two (concrete or abstract) charts $C1 = \langle a_1, a_2, \ldots, a_n \rangle$ and $C2 = \langle b_1, b_2, \ldots, b_n \rangle$. The concatenation of the two charts, denoted by $C1 +\!\!+ C2$, corresponds to the chart $\langle a_1, \ldots, a_n, b_1, \ldots, b_n \rangle$

We can also define a sub-sequence relationship among charts as defined below.

*Definition 2.4 (***Sub-sequence):** A chart (concrete or abstract) $C_1 = \langle e_1, \ldots, e_n \rangle$ is considered a *subsequence* of another chart $C_2 = \langle f_1, f_2, \ldots, f_m \rangle$ if there exist integers $1 \leq i_1 < i_2 < i_3 < i_4 \ldots < i_n \leq m$ where $e_1 = f_{i_1}$, $e_2 = f_{i_2}$, $\cdots$, $e_n = f_{i_n}$. We denote this relation by $C_1 \sqsubseteq C_2$. We also say that C_2 is a *super-sequence* of C_1.

An input trace can be viewed as a series of concrete events. To formalize we denote T as $\langle o_1, o_2, \ldots, o_{end} \rangle$ where each o_i is a concrete event from Σ_{con}. The ith event in a trace T is denoted by $T[i]$. In this paper, we consider a single trace, but the algorithm can be easily generalized to multiple traces.

Satisfaction of a chart follows the semantics of LSC. We refer to a sub-trace (or a segment of consecutive events in the trace) satisfying the chart C (either concrete or abstract) as an instance of C. To describe it simply, we use the following Quantified Regular Expressions (QRE) [28]. A quantified regular expression is very similar to standard regular expression with ';' as concatenation operator, '[-]' as exclusion operator (*i.e.* [-P,S] means any event except P and S) and * as the standard kleene-star.

*Definition 2.5 (***Instance of a Concrete Chart):** Given a concrete chart $C = \langle o_1, o_2, \ldots o_n \rangle$, a trace segment $SB = \langle sb_i, sb_{i+1}, \ldots, sb_{i+m-1} \rangle$ of a trace is an instance of C iff it is of the following QRE expression

$$o_1; [-o_1, \ldots, o_n]*; o_2; \ldots; [-o_1, \ldots, o_n]*; o_n.$$

Note the constraint involved in the definition of an instance of an object level LSC. In particular, we note the exclusion operations (*i.e.*, $[-o_1, \ldots, o_n]$) in Definition 2.5. This means, for $2 \leq i \leq n$, between occurrences of the events o_{i-1} and o_i in an instance of chart C, there should be no occurrences of events from the set $\{o_1, \ldots, o_n\}$. We refer to this constraint as the *instance constraint*.

An instance of a concrete/abstract chart in a trace T can be denoted by a pair (srt, end) where srt and end correspond to the starting and ending indices of the segment SB. The set of all instances of a chart C is denoted $Inst(C)$.

To illustrate the above definition of instances of a concrete chart, consider the following table consisting of 4 concrete traces.

ID	Trace
E1	$\langle a, b, c, b, a, d \rangle$
E2	$\langle a, b, b, a, d \rangle$
E3	$\langle a, f, b, e, a, f, d \rangle$
E4	$\langle a, b, e, a, x, d, g, a, b, c, a, d \rangle$

Consider the concrete chart $CC = \langle a, b, a, d \rangle$ and the table above. $E1$ is not an instance of CC due to event b appearing out of specified order, defined by CC, between the first occurrence of b and the second occurrence a. Similarly, $E2$ is not an instance of CC due to the event b duplicated between the first occurrence of b and the second occurrence a. $E3$ is an instance of CC since we ignore the occurrences of unrelated event f not specified in CC. Also, in $E4$ there are 2 instances of CC, namely $(1, 6)$ and $(8, 12)$. Note that we ignore the interleaving occurrences of unrelated events c, e, x, g not mentioned in CC.

We define the set of instances of an abstract chart AC to simply correspond to the union of instances of concrete charts CC whose projection equals to AC.

*Definition 2.6 (***Instance of an Abstract Chart):** Consider an abstract chart $AC = \langle a_1, a_2, \ldots, a_n \rangle$. Let the set $CCSet(AC) = \{cc | proj(cc) = AC\}$. Let $ObjIsts_{all} = \cup_{cc \in CCSet(AC)} \{(s', e') | (s', e') \in Inst(cc)\}$. The set of instances of AC is: $\{(s, e) | (s, e) \in ObjIsts_{all} \wedge (\nexists\ (s, f) \in ObjIsts_{all}.\ f < e)\}$

The definition simulates creating a monitor every time the first event of an LSC occurs. The next time a monitor state can be advanced on occurrence of a new event in the trace the state changes and it will look for the next event specified in the LSC. The condition $[(s, f) \in ObjIsts_{all}]$ in Defn. 2.6 ensures that (s, e) is a valid instance. The condition $[\nexists\ (s, f) \in ObjIsts_{all}.\ f < e]$ ensures that we capture the *first points* in the trace where the state of each monitor is updated to that when chart AC is satisfied.

*Definition 2.7 (***Trigger & Effect):** Trigger and effect are each defined as a chart. Trigger and effect can be either concrete or abstract chart.

In the context of this work, input trigger (or effect) corresponds to the pre-chart (or main-chart) of the LSC. The input trigger (or effect) and mined effect (or trigger) are composed to form an LSC L(trigger,effect). In this paper, we impose additional restriction requiring

the sets of events of the input chart and the output chart to be disjoint. This seems reasonable as most example LSC in the literature do not include repeated events. This restriction further improves the mining speed of the algorithm (see Sec. 4).

Witnesses and significant LSCs We consider the problem of computing the statistical values of a given chart. The definitions provided in this sub-section apply to both concrete and abstract charts. Given a concrete or an abstract LSC M and a trace T, we are interested in finding statistics denoting significance of M in T. To do so, we introduce the concepts of positive and negative witnesses.

A *positive witness* of a concrete or abstract LSC M = $L(pre,main)$, is a trace segment satisfying the $pre{+}{+}main$ chart – by extension the pre chart as well, since pre is a prefix of $pre{+}{+}main$. A *negative witness* of M is a positive witness of pre which cannot be extended to a positive witness of M (or $pre{+}{+}main$). We say that a negative witness is a *weak negative witness* (see discussion in [24]) if the positive witness of pre cannot be extended due to end-of-trace being reached.

We use the above notions of witnesses to define the statistical support and confidence metrics for LSC. Given a trace T, the *support* of an LSC M= $L(pre, main)$, 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 and weak-negative-witnesses of the LSC and the number of positive-witnesses of the LSC's pre-chart. Formally:

$$conf(L,T) \equiv_{def} \frac{|pos(L,T)|+|w\_neg(L,T)|}{|pos(pre,T)|}$$

Notation-wise, when T is understood from the context, it can be omitted. The confidence gives the measure of assurance that if a pre-chart is satisfied, either the main-chart is satisfied or the end of trace is reached.

The support metric is used to limit the extraction to frequently observed interactions. The confidence metric restricts mining to such pre-chart that is followed by a particular main-chart with high likelihood. We are especially interested in those LSCs with perfect confidence, that is with $conf(M) = 1$ (our algorithms, however, handle the case of $conf(M) < 1$ too; see Sec. 5).

Note that since we consider a given trigger (or effect), to return a non-empty result, mining must use a minimum support threshold that is at most equal to the number of instances of the given trigger (or effect) in the trace. Thus, the support threshold we use is in fact normalized according to the number of instances of the given trigger (or effect) in the trace (in our experiments we use $\sup(M) = 1$ as default). This seems a nice property, as both support and confidence threshold values are in the range of 0 to 1, and are hence intuitive for users to set and are comparable across different traces (we remark that this was not possible in [24], as there was no reference point to normalize to).

Problem statements The two problems of effect and trigger mining addressed in our work are stated below.

Problem Statement 1. Given a (concrete/abstract) trigger chart TGR, compute a complete set of (concrete/abstract) effect charts $EFCSet$, such that for each $EFC \in EFCSet$, the LSC $L(TGR,EFC)$ meets the minimum support and confidence thresholds in the set of input traces T.

Problem Statement 2. Given a (concrete/abstract) effect chart EFC, compute a complete set of (concrete/abstract) trigger charts $TGRSet$, such that for each $TGR \in TGRSet$, the LSC $L(TGR,EFC)$ meets the minimum support and confidence thresholds in the set of input traces T.

We refer to the minimum support and confidence thresholds as min_sup and min_conf. A trigger or effect whose resulting LSC meets the min_sup threshold is considered *frequent*. If it's corresponding LSC also meets the min_conf threshold, we refer to it as being *significant*.

In addition, in trigger mining (effects mining), we are only interested in those significant charts that are minimal (maximal). We want to get minimal triggers (maximal effects) that relate to the behavior described by the given effect (trigger). See the discussion on longest main charts and shortest pre charts in Sec. 5.

Statistical soundness & completeness An algorithm mining significant specifications is statistically sound and complete if all mined specifications are significant (sound), and all significant specifications are mined (complete). This notion is commonly used in data mining, and is guaranteed by, e.g., Daikon [9] or data mining applications (e.g., [21]). Our algorithms are sound and complete modulo the given traces and user-defined thresholds (our notion of soundness and completeness is thus independent of the quality of the traces used in terms of coverage etc., that is, in contrast to, e.g., [13]).

3. Mining Algorithm Outlines

Our mining algorithms take advantage of the following monotonicity property for concrete and symbolic LSCs (a proof is available in the technical report [22]).

*Property 1 (***Monotonicity - Positive Witness):** Given a (concrete/abstract) LSC M = $L(pre, main)$ and M' = $L(pre', main')$. If $pre{+}{+}main$ is a subsequence of $pre'{+}{+}main'$, every positive witness of M' is a positive witness of M. Hence, $sup(M') \leq sup(M)$.

The above property can be effectively used to prune the search space. For example, after ascertaining that the support of a (concrete or abstract) chart $L(a,b)$ is less than the minimum support threshold, one can prune the search space of all LSCs $L(pre,main)$ where $\langle a, b\rangle$ is a subsequence of $pre{+}{+}main$.

Our algorithm for mining effects given a trigger is outlined below.

Algorithm outline 1 (Effects Mining):

1) Consider an input trigger chart (abstract or concrete) TGR and input trace T. We compute $Inst(TGR, T)$.
2) We grow TGR by appending (abstract or concrete) events one-by-one in a depth-first, lexicographic order. We stop growing when the support of $TGR{+}{+}EVS$ (where EVS is a series of abstract events) is below $min\_sup \times |Inst(TGR, T)|$. We output $L(T,EVS)$ if its support and confidence are greater than the thresholds.
3) The result of the previous step corresponds to the set of all abstract effects. The input trigger TGR and each of the charts $TGR{+}{+}EVS$ form an LSC $L(TGR,TGR{+}{+}EVS)$, where EVS is the abstract effect mined.

A different approach is required for mining triggers given an effect, however. One cannot obtain instances of the given effect and grow them, in this case "backward", since this would find 'necessary triggers', that is, sequences without which the effect scenario never occurs (as in a past temporal logic expression of effect entails trigger). Although this may be interesting to compute, this is *not* what we are looking for. Rather, we are looking for 'sufficient triggers', and want the confidence value to capture the likelihood of the given effect to occur after the mined trigger. We outline the algorithm to do this below.

Algorithm outline 2 (Triggers Mining):

1) Consider an input effect chart (abstract or concrete) EFF and input trace T. We compute all charts $e{+}{+}EFF$, where e is a single event and $|Inst(e{+}{+}EFF)| \geq min\_sup \times |Inst(\text{EFF})|$. Let us refer to these single frequent event es as FEv – frequent single event triggers. From apriori property, all triggers whose support is above min_sup need to be composed by events in FEv.
2) Starting from a single-event frequent trigger G, recursively one grows a trigger G by adding one event from FEv at a time. At each step one need to check if we can create instances of $G{+}{+}e_n{+}{+}EFF$, where e_n is a new event.
3) If $|Inst(G{+}{+}e_n{+}{+}EFF)| < min\_sup$, from apriori property, charts of the form $G{+}{+}e_n{+}{+}EVS{+}{+}EFF$, where EVS is an arbitrary series of events, are infrequent (i.e., support is less than min_sup). Hence we do not need to grow the trigger $G{+}{+}e_n$ anymore.
4) For every G to be grown, $sup(G{+}{+}EFF) \geq min\_sup$. If $conf(L(G,EFF))$ is also greater than the minimum confidence threshold, we output LSC $L(G,EFF)$. If $L(G,EFF)$ is output there is no need to grow G anymore, since other triggers are non-minimal.

The algorithms perform post-processing to only return maximal effects and minimal triggers. An effect is maximal if there is no other significant effect that is a super-sequence of it. A trigger is minimal if there is no other significant trigger that is a sub-sequence of it. Additional details, including pseudo code, are given in the tech. report [22].

4. Experiments & Early Evaluation

To evaluate our work we created a prototype implementation of our algorithms. We used ApsectJ to monitor programs and generate traces of triplets (caller identifier, callee identifier, method signature). We report here on the results of some of our experiments (running on a Pentium M 1.6GHz 1.5GB RAM Windows XP Tablet PC). The algorithms are written using C#.Net compiled using VS.Net 2005. Complete results including traces are available from [22].

Below we consider two medium size third party open source applications; Jeti [4], a popular full featured instant messaging application, consisting of about 49K LOC, 3400 methods, 511 classes in 62 packages; and Columba [1], a rich email client application, consisting of 46K LOC, 6200 methods, 1139 classes in 226 packages. We refer to the traces generated from Jeti and Columba by J and MC respectively. All experiments reported below were conducted with $min\_sup = min\_conf = 1$.

Mining effects We provided 3 intuitive triggers to

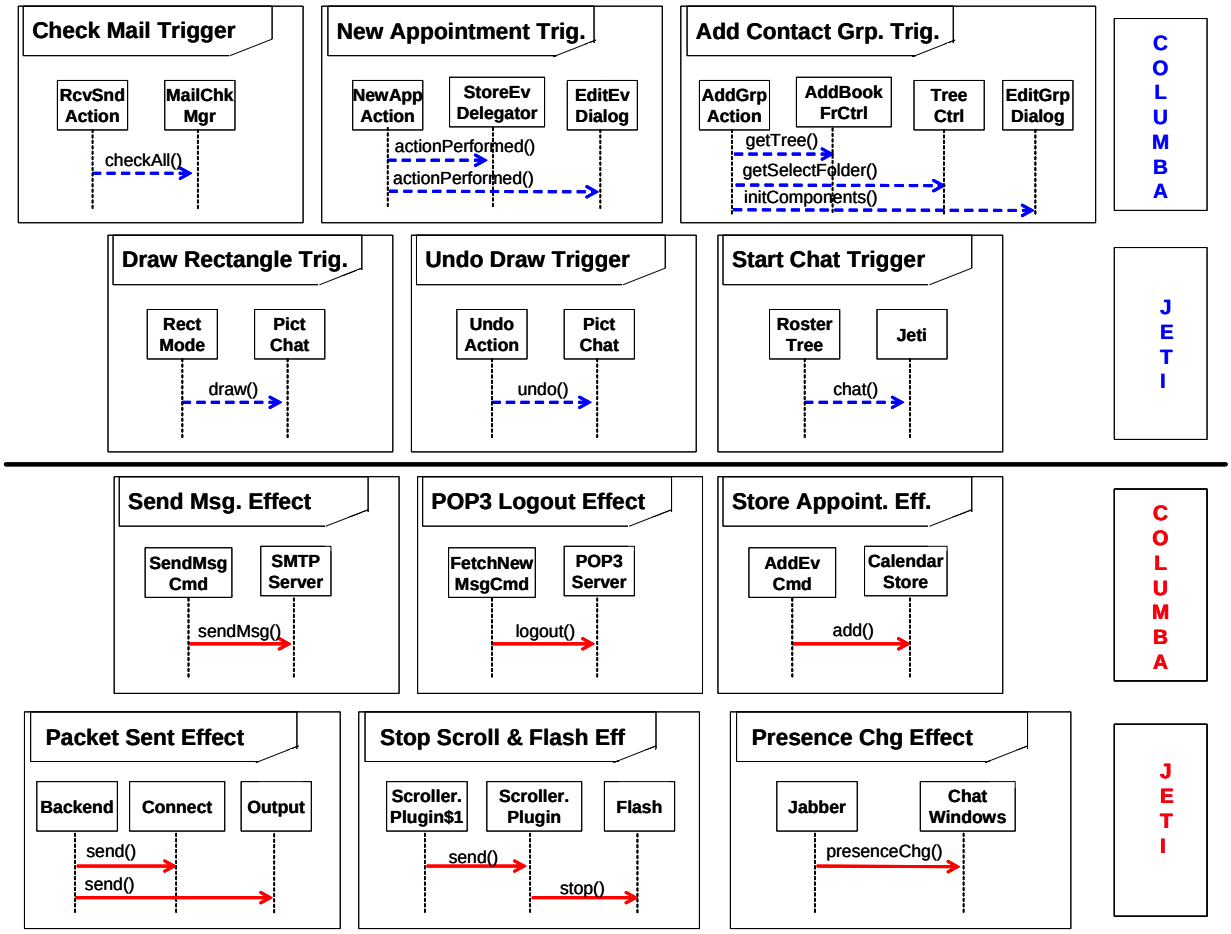


Figure 3. Triggers and Effects Used in Our Experiments

Exp.	Trace	Trigger Used	Time	|Mined|
MEJ1	J1	Start Chat	12.61s	5
MEJ2	J2	Draw Rectangle	0.55s	4
MEJ3	J2	Undo	0.23s	2
MEC1	CB	Check Mail	278.33s	11
MEC2	CB	New Appoint.	0.05s	1
MEC3	CB	Add Contact Grp	75.00s	2

Table 1. Experiments on Mining Effects

Exp.	Trace	Effect Used	Time	|Mined|
MTJ1	J1	Packet Send	0.13s	2
MTJ2	J2	Stop Scroll/Flash	0.94s	24
MTJ3	J1	Presence Chg	0.09s	29
MTC1	CB	Send Msg	17.47s	4
MTC2	CB	POP3 Logout	52.89s	32
MTC3	CB	Store Appoint.	135.72s	48

Table 2. Experiments on Mining Triggers

each application and analyzed the mined candidate specifications. Table 1 shows the experiments details. The columns (left to right) refer to the experiment id (e.g., MEJ1 = Mine Effect Jeti 1), trace id, trigger id, running time, and number of effects mined. The details of the triggers used are given in Fig. 3 (top).

One of the mined effects and its corresponding LSC are shown in Fig. 4(a). The trigger is an event called when the user asks to check for new mails in the server. The effect is of length 9. It describes the series of events that complete scenario of user checking email in the server. First, available email accounts (in our case only one account) are checked whether they need to be checked or not (events 1 & 2 in the effect). User can specify for an email account not to be checked. In this case, since we specify that the account is to be checked, the actual check is performed (event 3). Next, the local directory and the server are synchronized (events 5-7). Finally, downloaded files are deleted from the main server if the deletion feature is enabled (the methods are called in any case) (events 8-9).

Mining triggers We provided 3 intuitive effects to each application and analyzed the mined specifications. Table 2 shows the details of our experiments.

The details of the triggers used are given in Fig. 3 (bottom). Two of the mined triggers and their corresponding LSCs are shown in Figure 4(b). The input effect describes the actual event where a newly formed

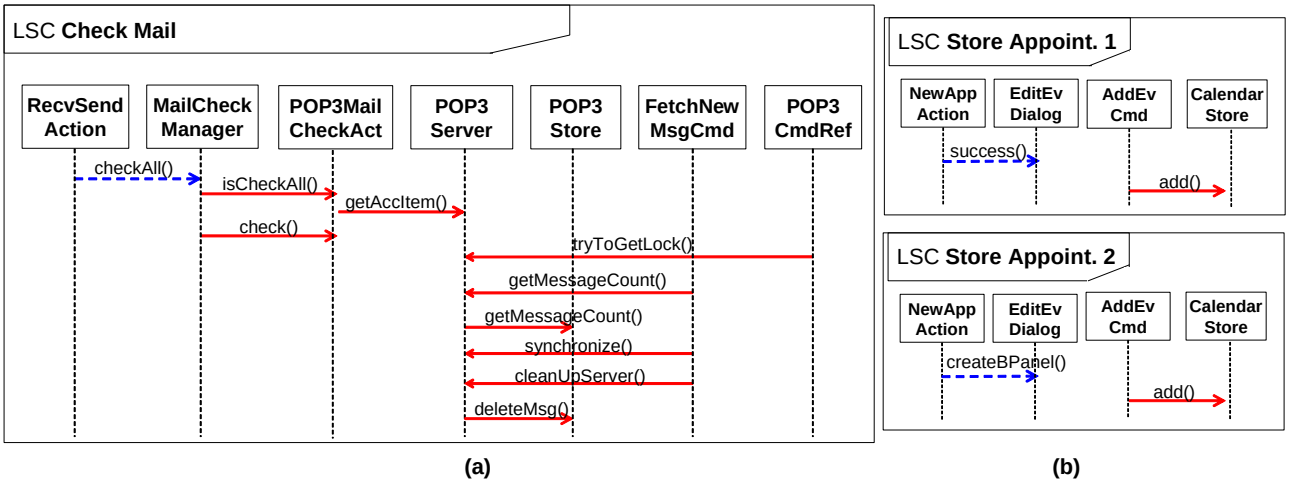


Figure 4. (a) Mine Effect: Check Mail; (b) Mine Triggers: Store Appointment

appointment is stored into the calendar, and we were interested in finding minimal triggers for this behavior. Two of these are shown in Fig. 4(b). One of the triggers is of `EditEventDialog`, when a user enters new appointment information `success()` method is called. Another trigger is when `EditEventDialog` needs to be created. In this case, we can note that regardless of the event dialog's "success" (i.e., user confirms the addition), the `add()` method of `LocalCalendarStore` is called.

Scalability To check the scalability of our work we generated traces of length 10K by chatting via Jeti, and then created longer traces using concatenation. We created 5 traces of length 1 to 5 million events, considered the effect and trigger used in experiments MTJ1 and MEJ1, and measured the running time of computing the corresponding triggers and effects over these long traces (Fig. 5). The graphs show that mining completes in reasonable time even when the trace is 5 million events long, and that running time grows only linearly with the input traces length.

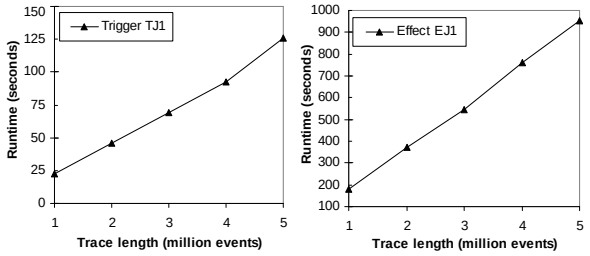


Figure 5. Scalability Experiments Results

Note that our original approach to mining a full set of LSCs, presented in [24], was much less scalable. While a formal comparison between the two work is impossible as the problems addressed are different, it is not surprising that trigger/effect mining is much more scalable, as the focus on the given trigger/effect significantly restricts the search space size (roughly, running time on equal traces was 400 times longer with the general approach (approx. 8100 sec. vs. 20 sec.)). The special case of scenario-based triggers and effects mining is thus indeed much more scalable and hence feasible for engineers' use.

Lessons learned While the above experiments show promising results in terms of the quality of scenarios mined and of the scalability of our approach, we have also learned some important lessons about the limitations and challenges faced by our current work, some of which are briefly listed below. First, results are most useful when only a small number of charts is mined, hence there is a need for additional filters (see the next section). Second, to best take advantage of the iterative interactive process, a methodology that integrates our mining approach with software testing and maintenance tasks using a supporting wizard-like tool is required. Third, we would like to get additional information beyond the temporal order of calls, hence there is a need to consider adding additional data, e.g., parameter values, to the traces, or combine with a miner mining value-based invariants like Daikon [9]. Finally, the visualization using sequence diagrams is important to the usability of our approach, as textual representations of scenarios are difficult to understand.

5. Advanced Features & Considerations

Less than perfect confidence While the most natural confidence value to set is 1, that is, requiring candidate

LSCs to hold as invariants on the input traces, our work supports setting up a lower confidence level. This may be appropriate in applications where tracing may output *imperfect traces* due to a lossy tracing mechanism [30], or may help exposing exceptions or bugs in the program under investigation (see, e.g., [6], [20], [30]). The support and confidence values for each mined LSC are included in the algorithms output. Thus, the option to set a less than perfect confidence level makes our work more robust and widens its applicability.

Long triggers Recall that given a trigger, our miner returns *longest* effects (main-charts) meeting the confidence threshold, as these are logically stronger post-conditions. Dually, given an effect, our miner looks for *shortest* triggers (pre-charts) meeting the confidence threshold, as these are logically strongest (following a 'weakest pre-condition' approach). Our experience (confirmed with experienced users of the Play-Engine tool [10]), shows that the latter, however correct, is not the best for all usages; longer triggers, however logically weaker, may provide the user with important additional information. Thus, we allow the user to choose mining of all triggers meeting the threshold, and presenting them sorted by their length.

Binding preserving abstraction Lifting the concrete mined scenarios up to class level abstract scenarios in a sound and complete way requires our algorithms to respect what we call *binding preserving abstraction* (BPA). That is, mining must abstract away concrete object identifiers while respecting the binding topology within a scenario. In this work we assume no two lifelines of the same LSC correspond to the same class, thus bypassing the need for BPA. We leave BPA for future work.

Additional user-guided filters and abstractions To better serve the user's needs in mining triggers and effects, and allow the user the use apriori knowledge of the program under investigation or knowledge acquired during previous mining, we offer an array of user-guided filters and abstractions. These include, e.g., *ignore set*, allowing the user to specify a set of method signatures to be ignored by the miner, and *consider set*, allowing the user to instruct the miner to only mine triggers (or effects) that include events from a specified set. Additional details may be found in [23].

Using the mined scenarios As mentioned earlier, our current work is a part of and an example for a larger framework we are working on, presenting and examining what we may call *user guided mining*, where spec mining is viewed as an interactive iterative process. In this regard, the end-to-end visual aspect of our work is an important usability factor. Both the input for the miner, trigger or effect sub scenarios, and the output of the miner, a set of LSCs, may be created and presented visually, within an industrial modeling tool (in our case, IBM RSA [3]), where they can be edited, manipulated, printed etc. Moreover, as shown in [24], the mined LSCs can be programmatically compiled into (monitoring) *scenario aspects* [26], serve as scenario-based tests for the application under investigation, and thus allow to 'validate' the mined LSCs during its subsequent executions.

6. Related Work

Automata-based specification mining Most specification miners produce an automaton (e.g., [5]–[7], [25]), and have been used for various purposes from program comprehension to verification. Unlike these, 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. Thus, 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., [11], [16], [29]). The specific mining of triggers and effects assists the user in focusing on investigating the aspects of the behavior of interest in the context of specific tasks.

Reverse engineering of sequence diagrams Many work suggest and implement different variants of reverse engineering of objects' interactions from program traces and their visualization using sequence diagrams (see, e.g., [2], [14]), which may seem similar to our current work. Unlike our work, however, all consider and handle only concrete, continuous, non-interleaving, and complete object-level interactions and are not using aggregations and statistical methods to look for higher level recurring scenarios; the reverse engineered sequences are used as a means to describe single, concrete, and relatively short (sub) traces in full (and thus may be viewed not only as concrete but also as 'existential'). In contrast, we look for universal (modal) sequence diagrams, which aim to abstract away from the concrete trace and reveal significant recurring potentially universal class-level abstract scenario-based specification, ultimately suggesting scenario-based system requirements.

Trigger querying in formal verification In [18], Kupferman and Lustig introduced and studied the problem of *trigger querying* in the context of formal

verification. Given a model M and a temporal formula ϕ, trigger querying is the problem of finding the set of scenarios that trigger ϕ in M. That is, if a computation of M has a prefix that follows the scenario, then its suffix satisfies ϕ. Our current work may be viewed as a special case of a dynamic analysis variant of trigger querying. We use (sub) scenarios to specify both triggers and effects. We investigate both trigger querying and its dual 'effect querying'.

7. Conclusion & Future Work

We presented scenario-based triggers and effects mining, as an interesting and useful special case of scenario-based specification mining in general. Experiments show promising results in terms of the quality of scenarios mined and of the scalability of our approach. The work is part of the larger framework of *user guided specification mining*, where specification mining is viewed as a task oriented iterative interactive process allowing the user to focus on issues of interest and use accumulated knowledge about the application under investigation. It aims to support property discovering tasks for debugging, evolution, runtime monitoring, and formal verification.

Based on the lessons learned listed above, future directions include extending our work to cover a larger subset of the LSC language (specifically, partial orders, conditions), making the prototype implementation available as a packaged tool, and conducting further experiments.

Acknowledgements We would like to thank David Harel for comments on our work on mining LSCs.

References

[1] "Columba, Java Email Client." http://www.columbamail.org/drupal/.

[2] "Eclipse Test and Performance Tools Platform," http://www.eclipse.org/tptp/.

[3] "IBM Rational Software Architect," http://www-306.ibm.com/software/rational/.

[4] "Jeti. Version 0.7.6 (Oct. 2006)." http://jeti.sourceforge.net/.

[5] M. Acharya, T. Xie, J. Pei, and J. Xu, "Mining API Patterns as Partial Orders from Source Code: From Usage Scenarios to Specifications," in *SIGSOFT FSE*, 2007.

[6] G. Ammons, R. Bodik, and J. R. Larus, "Mining Specification," in *POPL*, 2002.

[7] V. Dallmeier, C. Lindig, A. Wasylkowski, and A. Zeller, "Mining Object Behavior with ADABU," in *WODA*, 2006.

[8] W. Damm and D. Harel, "LSCs: Breathing Life into Message Sequence Charts," *J. on Formal Methods in System Design*, vol. 19, no. 1, pp. 45–80, 2001.

[9] M. Ernst, J. Cockrell, W. Griswold, and D. Notkin, "Dynamically discovering likely program invariants to support program evolution," *TSE*, vol. 27, no. 2, pp. 99–123, 2001.

[10] D. Harel and R. Marelly, *Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine*. Springer, 2003.

[11] D. Harel, "From play-in scenarios to code: An achievable dream." *IEEE Computer*, vol. 34, no. 1, pp. 53–60, 2001.

[12] D. Harel and S. Maoz, "Assert and Negate Revisited: Modal Semantics for UML Sequence Diagrams," *Software and Systems Modeling*, vol. 7, no. 2, pp. 237–252, 2008.

[13] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand, "Experiments on the effectiveness of dataflow- and control-flow-based test adequacy criteria," in *ICSE*, 1994.

[14] D. F. Jerding, J. T. Stasko, and T. Ball, "Visualizing Interactions in Program Executions," in *ICSE*, 1997.

[15] J. Klose, T. Toben, B. Westphal, and H. Wittke, "Check it out: On the efficient formal verification of Live Sequence Charts," in *CAV*, 2006.

[16] I. Krüger, "Capturing Overlapping, Triggered, and Preemptive Collaborations Using MSCs." in *FASE*, 2003.

[17] H. Kugler, D. Harel, A. Pnueli, Y. Lu, and Y. Bontemps, "Temporal Logic for Scenario-Based Specifications," in *TACAS*, 2005.

[18] O. Kupferman and Y. Lustig, "What triggers a behavior?" in *FMCAD*, 2007.

[19] M. Lettrari and J. Klose, "Scenario-Based Monitoring and Testing of Real-Time UML Models," in *UML*, 2001.

[20] D. Lo and S.-C. Khoo, "QUARK: Empirical assessment of automaton-based specification miners," in *WCRE*, 2006.

[21] D. Lo, S.-C. Khoo, and C. Liu, "Efficient mining of iterative patterns for software specification discovery," *SIGKDD*, 2007.

[22] D. Lo and S. Maoz, "Mining scenario-based triggers and effects (tech. rep. version and additonal details)," *Technical Report avail at. www.comp.nus.edu.sg/∼dlo/trigeff/*, 2008.

[23] D. Lo, S. Maoz, and S.-C. Khoo, "Mining modal scenarios from program execution traces," Dept. of Computer Science, National University of Singapore, Tech. Rep. TRC8/07, 2007.

[24] ——, "Mining Modal Scenario-Based Specifications from Execution Traces of Reactive Systems," in *ASE*, 2007.

[25] D. Lorenzoli, L. Mariani, and M. Pezzè, "Automatic Generation of Software Behavioral Models," in *ICSE*, 2008.

[26] S. Maoz and D. Harel, "From multi-modal scenarios to code: compiling LSCs into AspectJ," in *SIGSOFT FSE*, 2006.

[27] R. Marelly, D. Harel, and H. Kugler, "Multiple Instances and Symbolic Variables in Executable Sequence Charts," in *OOPSLA*, 2002.

[28] K. Olender and L. Osterweil, "Cecil: A sequencing constraint language for automatic static analysis generation." *IEEE TSE*, vol. 16, pp. 268–280, 1990.

[29] S. Uchitel, J. Kramer, and J. Magee, "Detecting implied scenarios in message sequence chart specifications." in *SIGSOFT FSE*, 2001.

[30] J. Yang, D. Evans, D. Bhardwaj, T. Bhat, and M.Das, "Perracotta: Mining temporal API rules from imperfect traces." in *ICSE*, 2006.