Related Work

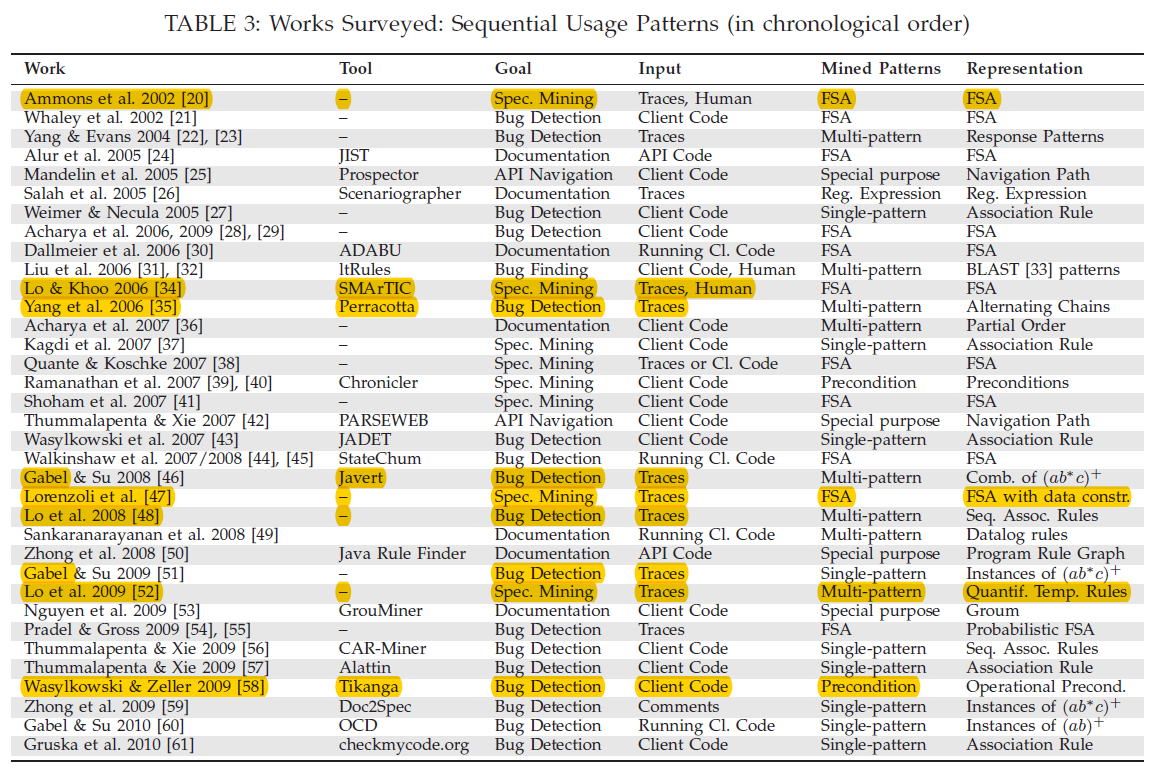
1. Who made specification mining  
   “There have been specification mining approaches over the last years [1,2,3,…]”
2. Static vs. Dynamic
   1. Static
      1. Ramanathan et al.
         1. Path-sensitive inference of function precedence protocols
         2. Static specification inference using predicate mining
      2. Shoham et al.
         1. Static specification mining using automata-based abstraction
   2. Dynamic
3. The problem is NP hard
   1. [3]
4. API usage
5. Model/Automaton-based

[3]  
kTail algorithm: finite state model => more compact one  
Approaches that leverage kTail to infer models without developer supervision

[30]  
There have been numerous work in the research of automoaton-based specification mining [ … ]  
A et al. have …. [x]  
B et al. provided… [x]

[19] SMArTIC  
There have been numerous work in the area of specification mining. They can be classified into two groups, depending on how the mined specifications are represented: automaton-based [ … ] and non-automaton based [ … ] specification mining.

[Periodic Task Mining in Embedded Systems]  
formalisms including state machines [3] [39], Petri nets [42], various types of invariants [21] [19] and UML models [29] [2]. Some formalisms support timing, such as mining LTL expressions [33], CTL expressions [6], or hybrid system automata [37].



**Yang et al. (*Perracotta, “Mining Temporal API Rules from Imperfect Traces”*)**

mines two-event temporal rules from execution traces. To infer these rules, Perracotta uses a set of predened rule templates and partitions input traces to several sub-traces. It computes satisfaction rate of a template, which is the number of partitions satisfying the template divided by the number of total partitions

[Gecco] A simple heuristic to explore the space of possible patterns is to search for specific pattern templates. In this work, simple two-events patterns are mined from the execution traces. These patterns follow the template event1 always followed by event2. Such simple patterns, once mined, can be combined to produce larger patterns.

[Rule-based specification mining leveraging learning to rank] that mines two-event temporal rules from execution traces. To infer these rules, Perracotta uses a set of predefined rule templates and partitions input traces to several sub-traces. It computes satisfaction rate of a template, which is the number of partitions satisfying the template divided by the number of total partitions

[10] mines several two variable response patterns and chains alternating properties together to form multi-variable properties. It also supports approximate inference to filter out uninteresting properties

[11] adopted a similar approach for locating alternating events, Perracotta, that introduced novel methods for handling imperfect traces. Sources of imperfection include interleaved concurrent executions, omitted information (like memory addresses), or bugs. In this work, the authors briefly describe a heuristic for combining simple alternating patterns, but the approach is limited to finding simple sequencing patterns

**Lo D. et al. (*SMArTIC-2006, SpecForge, “Mining […] SW Maintenance”-2008)***

[Gecco] stated by Lo et al., such combination methods might miss some multi-event patterns or create ones that are not meaningful. Alternatively, Lo et al. generalize the two-event template to a multi-event template, and mine patterns in the form of whenever a series of events occurs, eventually another series of events will occur.

[Rule-based specification mining leveraging learning to rank] 2008 extend Yang et al.’s approach by inferring from execution traces temporal rules with arbitrary lengths instead of two-event rules;  
SMArTic that applies a variant of k-tails automaton learning algorithm to infer finite state automatons (FSAs) from a set of execution traces;  
SpecForge that synergizes different FSA-based specification miners by introducing novel concepts of model fission and model fusion

**Lemieux et al. (*Texada, “General LTL Specification Mining”*)**

introduce Texada that mines temporal specifications in the form of linear temporal logic (LTL) of arbitrary length and complexity

[Gecco] mentioned approaches cannot mine complex patterns such as When event1 occurs, either event2 occurs just after and event3 will never occur after, or event2 will not occur just after and event3 will eventually occur. Lemieux et al. [1] proposed a model-checking-base tool, Texada, to mine such complex patterns. However, the pattern templates used for the search have to be specified beforehand. In other words, it is impossible to find what you are not explicitly looking for. This precludes exploratory searching, which can reveal previously unanticipated patterns

[Rule-based specification mining leveraging learning to rank] introduce Texada that mines temporal specifications in the form of linear temporal logic (LTL) of arbitrary length and complexity

**Ernst et al. (*Daikon, “Dynamically Discovering Likely Program Invariants”*)**

[4] Daikon is a tool that automatically infers likely program invariants using statistical inference from a program's execution traces.

[6] proposed automatic deduction of formal specifications. Their Daikon tool works by learning likely invariants involving program variables from dynamic traces.

[11] is a dynamic technique that is similar in spirit to our own analysis. Daikon locates invariants on the values of variables, while we locate invariants on the sequencing of function invocations

**Gabel and Su (*Javert, “Fully Automatic Mining of General Temporal Properties from Dynamic Traces”*)**

[Gecco] combination (“chaining”) strategy was also proposed by Gabel and Su

[10] also mines alternating patterns, along with resource ownership patterns (i.e., (ab\*c)\*) and composes them into more complex properties

**Dwyer (“Property Specification Patterns for Finite-State Verification”)**

**Ammons (*“Mining Specifications”*)**

[11] first characterized the inference of temporal specifications as a language learning problem. In this work, the authors used a probabilistic finite automaton learner to extract likely specifications. A key challenge with their approach was simplifying specifications to an acceptable level of precision

[12] develop a specification miner, Strauss, that mines specification by learning a probabilistic finite state automaton. Unlike our approach, Strauss requires the input alphabet of the automaton to be specified, but it does have the potential to find more complex specifications.

**Li (“*Scalable Spec Mining for Verification and Diagnosis” – 1st Lady w/ hammer”*)**

extend Yang et al.'s work by extracting simple linear temporal logic (LTL) rules from execution traces for hardware design

[Rule-based specification mining leveraging learning to rank] also extend Yang et al.’s work by extracting simple linear temporal logic (LTL) rules from execution traces for hardware design

[10] extend Perracotta to mine simple LTL patterns from traces and merge these to analyze digital circuits. These same LTL patterns are mined between data invariants in [4], [8].

**Beschastnikh et al. (Synoptic, “”)**

[Rule-based specification mining leveraging learning to rank] automatically mines three types of temporal rules from execution traces and uses them to generate a concise finite state automaton (FSA) model that satisfies these rules

**Engler et al.**

* Static Inference / Specification Mining
* [11] irst introduced the idea of matching an alternating pattern over a program to produce possible specification candidates. This approach suffered from imprecision, so the authors used statistical methods to rank the possible properties

**Ramanathan**

[11] static analysis [24] for detecting “function precedence protocols.” These specifications are of the form “function x is called on all paths leading to an invocation of function y.” The authors later generalized this technique to include other predicates like constraints on variables [25]. These specifications are of limited expressiveness; they correspond to the simple pattern (b+a).

**GKTail**

[11] Combining the ideas of invariant detection and temporal property mining, Lorenzoli et al. have developed a dynamic analysis algorithm for extracting software behavioral models [20]. The algorithm, GK-tail, builds an Extended Finite State Machine from a set of dynamic traces. The transitions in these extended models include both a called function or method and a set of constraints on the parameters or environment.

**About Static**

[11] Various static analyses [3,16,30] take as input a type and produce as output an automaton that encodes legal call sequences of operations on that type. Call sequences are considered legal if they do not lead to an assertion failure or another exceptional control path.