Mining Past-Time Temporal Rules From Execution Traces

David Lo[1,2] Siau-Cheng Khoo[2] Chao Liu[3]

[1]Singapore Management University
[2]National University of Singapore
[3]Microsoft Research, Redmond

Issue on Software Specifications

- Documented specifications are often lacking, poor, outdated and incomplete

  Hard deadlines & `short-time-to-market'

  Productivity == LOC or completed project

  High turn-over rate of IT professionals

  Difficulties & programmer's reluctance in writing formal specs
  (Ammons et al., POPL'02, Yang et al., ICSE'06)

The Specification Problem

- **Contributes to high software costs**
  - Program comprehension = 50% of maintenance cost
  - High maintenance cost = 90% total cost
    (Erlikh, 2000; Cimitile & Canfora, 2001)
  - US GDP software component = 214.4 billion USD.

- **Causes challenges in ensuring correctness of systems**
  - Difficulty in verifying correctness of systems
  - US National Institute of Standards and Technology
    59.5 Billions annual lost due to bugs

Specification Mining (SM)

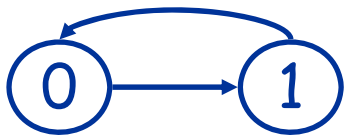
A process to discover protocols that a code exhibit, often through an analysis of its execution traces (ABL02 [POPL])

Benefits:

Aid Program Comprehension and Maintenance

Aid Program Verification

Automaton-based SM



RR01 [ICSE], CW98 [TOSEM]

ABL02 [POPL], AMBL03 [PLDI],

WML02 [ISSTA] , AXPX07 [FSE]

MP05 [ICEECS], LK06 [FSE]

Rule-based SM

<Lock> -> <Unlock>

YEBBD06 [ICSE]

LKL08 [DASFAA,JSME]

Only **future-time** temporal rules are mined

Past-Time Temporal Rules

Whenever a series of events *pre* occurs, previously, another series of events *post* happened before, denoted as: pre $->_P$ post

Among most-widely used temporal logic expressions (Dwyer,ICSE'99)

Why Important ?

- Past-time temporal exp. -> complex future-time temporal exp. (Laroussinie et al., TCS'95, LICS'02)
- Not minable by existing algorithms mining future time rules (Yang et al. ICSE'06, Lo et al. JSME'08]
- Many interesting properties are more intuitively expressed in past-time
- Many interesting properties are non-symmetric

Sample Past-Time Rules

- Whenever a file is used (read or written), it needs to be opened before.

  $$file\_used \rightarrow_p file\_open$$

- Whenever SSL_read is performed, SSL_init needs to be invoked before.

  $$ssl\_read \rightarrow_p ssl\_init$$

- Whenever a valid client request a non-sharable resource and the resource is not granted, previously the resource had been allocated to another client that requested it.

  $$request, not\_granted \rightarrow_p request, grant$$

- Whenever money is dispensed from an ATM, previously, card was inserted, pin was entered, user was authenticated and account balance suffices.

  $$dispense \rightarrow_p card, pin, authenticate, balance\_suffice$$

Outline

- Motivation and Introduction
- Concepts
  - Past-Time LTL, Statistical Significance
  - Soundness and Completeness
- Mining Past-Time Rules
  - Mining Strategy, Pruning Properties
  - Removal of Redundant Rules
  - Mining Framework
- Preliminary Experiments
- Discussion
- Related Work
- Conclusion & Future Work

Concepts

Past-Time Linear Temporal Logics (PLTL)

- Linear Temporal Logic (LTL)
  - Logic that works on program paths
  - A path corresponds to an execution trace
- Past-Time Linear Temporal Logic (PLTL)
  - Add LTL with past time operators
  - More succinct than LTL
- Temporal operators under consideration
  - `G' – Globally
  - `F' – Once in the future
  - `X' – Next (immediate)
  - `F^{-1}' – Once in the past
  - `X^{-1}' – Previous (immediate)

PLTL- Examples

- $X^{-1}F^{-1}$ (file_open)

  Meaning: At a time in the past file is opened

- G(file_read -> X^{-1} F^{-1} (file_open))

  Meaning: Globally whenever file is read, at a time in the past file is opened

- G((account_deducted ^ XF (money_dispensed)) -> ($X^{-1}F^{-1}$ (balance_suffice ^ ($X^{-1}F^{-1}$ (cash_requested ^ ($X^{-1}F^{1}$ (correct_pin^($X^{-1}F^{-1}$ (insert_debit_card)))))))))

  Meaning: Globally whenever one's bank account is deducted and money is dispensed (from an ATM), previously user inserted debit card, entered correct pin, requested for cash and account balance suffices.

Notations and Scope of Mined Rules

- Denote a past-time rule as pre ->$_P$ post
- Sample mappings btw. rule representations and PLTL expressions

Notation	LTL Notation
$a \hookrightarrow_P b$	$G(a \rightarrow X^{-1}F^{-1}b)$
$\langle a, b\rangle \hookrightarrow_P c$	$G((a \wedge XFb) \rightarrow (X^{-1}F^{-1}c))$
$a \hookrightarrow_P \langle b, c\rangle$	$G(a \rightarrow X^{-1}F^{-1}(c \wedge X^{-1}F^{-1}b))$
$\langle a, b\rangle \hookrightarrow_P \langle c, d\rangle$	$G((a \wedge XFb) \rightarrow (X^{-1}F^{-1}(d \wedge X^{-1}F^{-1}c)))$

- Scope of minable temporal expressions

$$
\begin{aligned}
rules &:= G(pre \rightarrow post) \\
pre &:= (event) | (event \wedge XF(pre)) \\
post &:= (event) | (event \wedge X^{-1}F^{-1}(post))
\end{aligned}
$$

Statistical Significance Metrics

- Distinguish Significant Rules via Statistical Notions
  - Support: The number of traces supporting the premise *pre*
  - Confidence: The likelihood of the premise *pre* being preceded by the consequent *post*

Sample Traces

Identifier	Trace/Sequence
S1	⟨c, b, a, e, b, a⟩
S2	⟨c, b, e, a, e, b, c, a⟩
S3	⟨d, a⟩

Rule: $\langle b,a \rangle \rightarrow_p \langle c \rangle$

Support: 2

Corres. to S1 and S2

Confidence: 100%

All occurences of <b,a> is preceded by <c>

Rule: $\langle b,a \rangle \rightarrow_p \langle e \rangle$

Support: 2

Confidence: 50%

Soundness and Completeness

- Ensure Soundness and Completeness
  - With respect to input traces and specified thresholds

- Sound

  All mined rules are statistically significant

- Complete

  All statistically significant rules are mined or represented

- Commonly used in data/pattern mining

Mining Past Time Temporal Rules

High-Level Mining Strategy

- Mining Option 1: Check for all 2-event rules (n x n of them) for statistical significance.
  - Not scalable for rules of arbitrary lengths.

- Our Mining Strategy: Consider mining as a search-space traversal for significant rules
  - Explore the search space depth-first
  - Identify significant rules

- Employ pruning strategies to throw away search space containing insignificant rules
- Detect search spaces containing redundant rules early during the mining process

Anti-Monotone Pruning Strategies

$$\frac{[Apriori - Support] \quad Rx = p \rightarrow_{\rho} c;\ Ry = q \rightarrow_{\rho} c \quad p \sqsubset q \quad sup(Rx) < min\_sup}{sup(Ry) < min\_sup} \qquad Ry \text{ is not significant}$$

$$\frac{[Apriori - Confidence] \quad Rx = p \rightarrow_{\rho} c;\ Ry = p \rightarrow_{\rho} d \quad c \sqsubset d \quad conf(Rx) < min\_conf}{conf(Ry) < min\_conf} \qquad Ry \text{ is not significant}$$

Rx: a ->$_p$ z ; sup(Rx) < min_sup

Ry$_s$

- a,b ->$_p$ z
- a,b,c ->$_p$ z
- a,c ->$_p$ z
- a,b,d ->$_p$ z
- ….

} **Non-significant**

Rx: a ->$_p$ z ; conf(Rx) < min_conf

Ry$_s$

- a ->$_p$ z,b
- a ->$_p$ z,b,c
- a ->$_p$ z,c
- a ->$_p$ z,b,d
- ….

} **Non-significant**

Detecting Redundant Rules

$$\frac{Rx = p \rightarrow_P c;\ Ry = q \rightarrow_P d \quad p{+\!+}c \sqsubset q{+\!+}d \quad sup(Rx) = sup(Ry) \quad conf(Rx) = conf(Ry)}{Rx\ is\ redundant}$$

Redundant rules are identified and removed early during mining process.

Rx: a ->$_P$ b,c,d

Ry$_s$

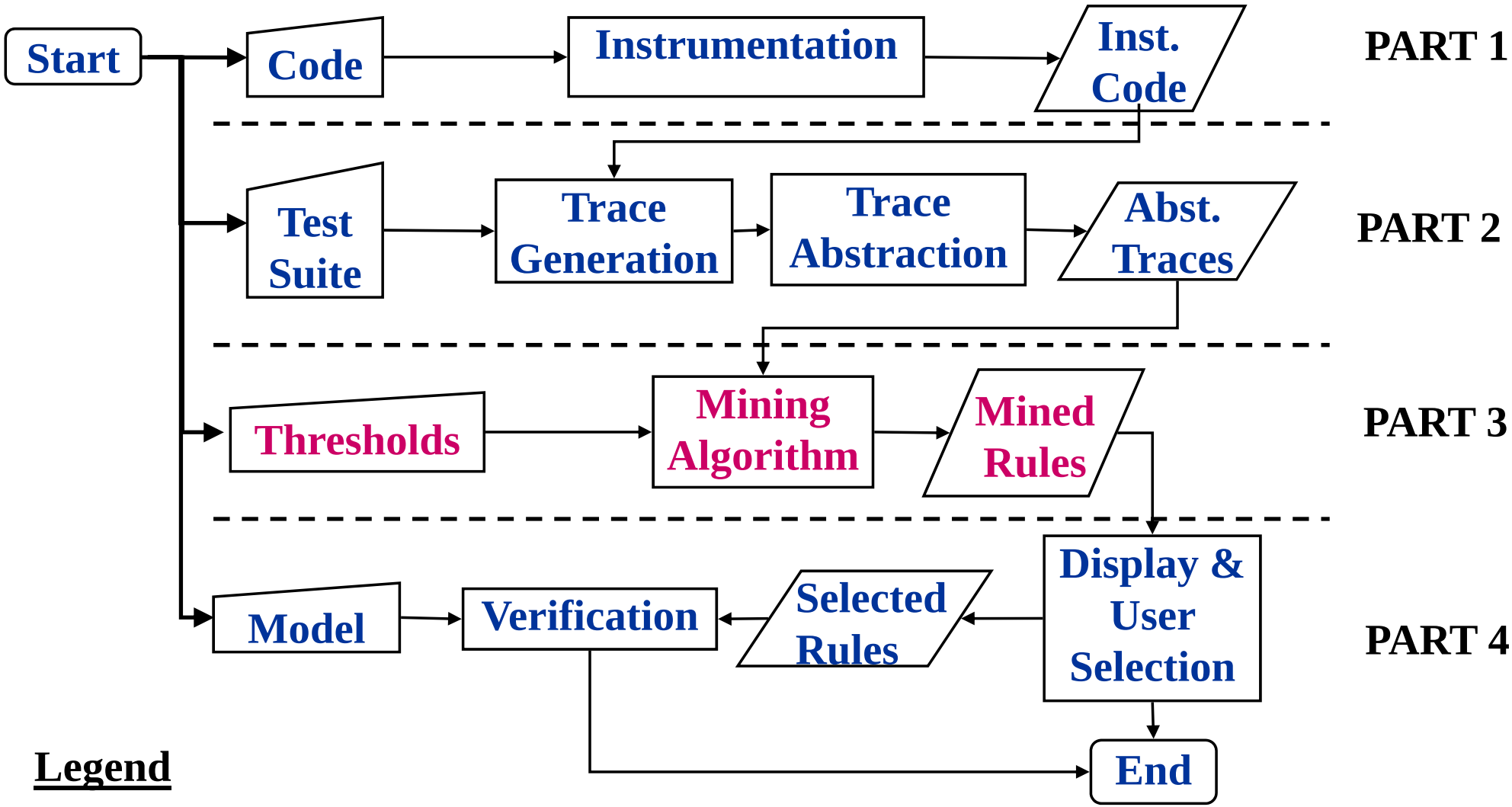
a ->$_P$ b
a ->$_P$ c
a ->$_P$ b,c
a ->$_P$ b,d
….

Redundant iff sup and conf are the same

Algorithm Steps

- Step 1: Generate a pruned set of significant pre-conditions satisfying the minimum support threshold.
- Step 2: For each pre-condition, find occurrences of pre in the trace database.
- Step 3: For each pre-condition, generate a pruned set of significant post-conditions satisfying the minimum confidence threshold.
- Step 4: Remove remaining rules that are redundant. Note that many/most redundant rules have been removed at step 1 and 3.

Mining Framework



Preliminary Experiments

Experiment Setups – JBoss Application Server

- JBoss Application Server (JBoss AS)
  - One of the most widely used J2EE application server
  - Analyze the transaction and security component
- Program Instrumentation & Trace Generation
  - Instrument the application using JBoss-AOP
  - Run regression tests from JBoss AS distribution
- Transaction component
  - 2551 events, 64 unique events
  - min_sup: 25, min_conf: 90%
  - Mining time: 30 seconds , Mined non-redundant rules: 36
- Security component
  - 4115 events, 60 unique events
  - min_sup: 15, min_conf: 90%
  - Mining time: 2.5 seconds, Mined non-redundant rules: 4

A Rule from JBoss Transaction

Premise	$\rightarrow_P$	Consequent
TransactionImpl.isDone()		TxManagerLocator.getInstance() TxManagerLocator.locate() TxManagerLocator.tryJNDI() TxManagerLocator.usePrivateAPI() TxManager.getInstance() TxManager.begin() XidFactory.newXid() XidFactory.getNextId() XidImpl.getTrulyGlobalId() TransImpl.assocCurrentThread() … 5 events … TxManager.getTransaction()

Whenever a transaction is checked for completion (premise), previously transaction manager is located (ev 1-4 consequent), transaction manager & impl are initialized (ev 5-6,10-12), ids are acquired (ev 7-9,13-15) and transaction object is obtained from the manager (ev 16).

A Rule from JBoss Security

Premise	$\xrightarrow{P}$	Consequent
SimplePrincipal.toString() SecAssoc.getPrincipal() SecAssoc.getCredential() SecAssoc.getPrincipal() SecAssoc.getCredential()		XLoginConfImpl.getConfEntry() PolicyConfig.get() XLoginConfImpl$1.run() AuthenInfo.copyAppConfEntry() AuthenInfo.getName() ClientLoginModule.initialize() ClientLoginModule.login() ClientLoginModule.commit() SecAssocActs.setPrincipalInfo() SetPrincipalInfoAction.run() SecAssocActs.pushSubjectContext() SubjectThreadLocalStack.push()

Whenever principal and credential info is required (the premise), previously config. info is checked to determine the auth. service availability (ev 1-5), actual authentication events are invoked (ev 6-8) and principal info is bound to the subject (ev 9-12)

Discussions

- Setting min-sup/conf threshold
  - Appropriate values depend on application
  - Mining as an iterative process
- Sound and Complete
  - With respect to trace and specified thresholds
  - If trace is not complete or buggy so does the results
  - Confidence provide a measure of tolerance to buggy traces
- Scalability
  - Algorithm works better with many shorter traces than one very long trace
  - It's better to split a trace to sub-traces
    - Focus on immediate inter-component interaction (Mariani et al., ICSE'08)
    - Trace abstraction (Ammons et al., POPL'02)

Related Work

- Daikon
  - Complement Daikon by mining temporal constraints
- Mining Automata
  - Many work: ABL02, RR01, MP05, AXPX07, LK06, …
  - Diff: Focus on statistically significant property rather than overall behavior
- Mining Future-Time Temporal Rules
  - Many work: YEBBD06, LKL08, …
  - Diff: Mining past-time temporal rules
- Mining Sequence Diagram: BLL06, LMK07, LM08, ...
- Mining from Code: RGJ07, WN05, …
- Data Mining: S99, AS94, YHA03, WH04, LKL07, …

Conclusion

- Propose a new approach to mine past-time temporal rules using dynamic analysis, not minable by existing tools:

> Whenever a series of events *pre* occurs, previously, another series of events *post* happened before, denoted as: pre $->_P$ post

- Address the problem of runtime costs by employing smart pruning strategies.
  - Throw away insignificant rules en-masse
  - Throw away redundant rules en-masse
- Preliminary experiments on traces of JBoss AS show utility of the technique to discover program behavioral rules/specifications

Future Work

- User Guided Mining
  - Let user provide more information to the mining process aside from the significance thresholds
  - Mining Scenario-Based Triggers and Effects (- ASE'08 – to-appear) – Mining Sequence Diagram
- Mining more complex LTL expressions
  - Incorporating both future and past-time temporal rules
- Improving the scalability of the technique
  - Abstraction technique
  - Pruning strategies
- Experimenting with more case studies

Thank you

Comments ? Questions ? Advices ?