Declarative Specification of FSM-Inference Algorithms

Observations → InvariMint → Model

Declarative specification
Applications:

- Mental model validation
- Test case generation
- Anomaly detection
- Log summarization

Cook et al. TSE 1999
Mariani et al. ICSE 2007
Dallmeier et al. ASE 2009
Beschastnikh et al. FSE 2011
Applications:

- Mental model validation
- Test case generation
- Anomaly detection
- Log summarization

Cook et al. TSE 1998
Mariani et al. ICSE 2007
Dallmeier et al. ASE 2009
Beschastnikh et al. FSE 2011
**FSM-inference**

**Applications:**
- Mental model validation
- Test case generation
- Anomaly detection
- Log summarization

---

**Observations**

---

**Model**

---

**Invalid**

---

**Read Only**

---

**Write**

---

---

**Cook et al. TSE 1998**

---

**Mariani et al. ICSE 2007**

---

**Dallmeier et al. ASE 2009**

---

**Beschastnikh et al. FSE 2011**
Prior work:

- Cook et al. TSE 1998
- Mariani et al. ICSE 2007
- Dallmeier et al. ASE 2009
- Beschastnikh et al. FSE 2011

Model inference

- Specification mining
- Process discovery
- Grammar inference

Observations

Ivan Beschastnikh
University of Washington
FSM-inference in prior work

Observations

Prior work:
- Cook et al. TSE 1998
- Mariani et al. ICSE 2007
- Dallmeier et al. ASE 2009
- Beschastnikh et al. FSE 2011

Inference algorithm

Invalid

Read Only

Write
FSM-inference representations

Observations

Inference algorithm

Model

Invalid
Read Only
Write
FSM-inference representations

Observations

Code

Model

Invalid

Read Only

Write

/** Implements the KPar algorithm as defined in Blomer & Feldman ’92. */
public class KPar {
    static {
        Logger = Logger.getLogger("KPar");
    }

    /** Constructs and returns a PartitionGraph by applying KPar to g,
     * @return a new PartitionGraph created by applying KPar to g.
     */
    public static PartitionGraph kPar(PartitionGraph g, int k) {
        assert g != null; // Use a safe, null-safe version of g instead.
        return g.kPar(k);
    }

    /** Finds all possible merges in g. Requires making a new call to
     * @return a new PartitionGraph created by applying KPar to g.
     */
    private static void attemptMerge(PartitionGraph g, int k) {
        Set<PartitionMultiMerge> mergers = new LinkedHashSet<PartitionMultiMerge>();
        // Keeps track of the merges that we want to perform.
Formally, given two sequences of length $k$ $\text{seq}_1 = (x^1_1, P^1_1) \ldots (x^1_k, P^1_k)$ and $\text{seq}_2 = (x^2_1, P^2_1) \ldots (x^2_k, P^2_k)$, we say that

- $\text{seq}_1 = \text{seq}_2$ if $\forall i = 1, \ldots k$ $x^1_i = x^2_i$ and $P^1_i \equiv P^2_i$
- $\text{seq}_1 \subseteq \text{seq}_2$ if $\forall i = 1, \ldots k$ $x^1_i = x^2_i$ and $P^1_i \Rightarrow P^2_i$

Given two $k$-futures $f1 = \{\text{seq}^1_1, \ldots, \text{seq}^1_n\}$, with $\text{seq}^1_i = (x^1_i, P^1_i) \ldots (x^k_i, P^k_i)$ and $f2 = \{\text{seq}^2_1, \ldots, \text{seq}^2_n\}$, with $\text{seq}^2_i = (x^1_i, P^1_i) \ldots (x^k_i, P^k_i)$

- $f1$ is equal to $f2$ if $\exists j = 1, \ldots n_1 \exists j = 1, \ldots n_2 \exists i = 1, \ldots n_1 \exists i = 1, \ldots n_2 \ s.t. \ \text{seq}^1_i \equiv \text{seq}^2_i$

- $f1$ weakly subsumes $f2$ if $\forall i = 1, \ldots n_1 \exists j = 1, \ldots n_2 \ s.t. \ \text{seq}^1_i \subseteq \text{seq}^2_j$, and vice versa $\forall i = 1, \ldots n_1 \ s.t. \ \text{seq}^2_i \subseteq \text{seq}^1_i$

- $f1$ strongly subsumes $f2$ if $\forall j = 1, \ldots n_2 \exists i = 1, \ldots n_1 \ s.t. \ \text{seq}^2_j \subseteq \text{seq}^1_i$
Formally, given two sequences of length $k$ as $seq_1 = (s_1, P_1^1) \ldots (s_k, P_k^1)$ and $seq_2 = (s_1, P_1^2) \ldots (s_k, P_k^2)$, we say that

- $seq_1 \sqsubseteq seq_2$ if $\forall i = 1 \ldots k$, $s_i = s_i$ and $P_i^1 \equiv P_i^2$
- $seq_1 \subseteq seq_2$ if $\forall i = 1 \ldots k$, $s_i = s_i$ and $P_i^1 \Rightarrow P_i^2$

Given two $k$-futures $f_1 = \{seq_1, \ldots, seq_{n_1}\}$, with $seq_1 = (s_1^1, P_1^1) \ldots (s_k^1, P_k^1)$ and $f_2 = \{seq_2, \ldots, seq_{n_2}\}$ with $seq_2 = (s_1^2, P_1^2) \ldots (s_k^2, P_k^2)$

- $f_1$ is equivalent to the class $EFSM$ if $\forall j = 1 \ldots n_1$
- $f_1$ weakly subsumes $f_2$ if $n_1 = n_2$, $\forall j = 1 \ldots n_1 \exists i = 1 \ldots n_1$ s.t. $seq_i^2 \subseteq seq_i^1$, and vice versa $\forall i = 1 \ldots n_1 \exists j = 1 \ldots n_1$ s.t. $seq_j^2 \subseteq seq_j^1$
- $f_1$ strongly subsumes $f_2$ if $\forall j = 1 \ldots n_2 \exists i = 1 \ldots n_1$ s.t. $seq_j^2 \subseteq seq_i^1$

**Input:** event log $L$

1. Let $initialGraph = extract(L)$
2. Let $I = mineInvariants(initialGraph)$
3. Let $(V, E) = partition(initialGraph)$
4. While $(V, E)$ does not satisfy invariants $I$
5. // $p$: event → boolean, $\pi$: partition that will be split
6. Let $(p, \pi) = selectSplit((V, E), I)$
7. Let $\pi_1 = \pi$
8. Let $\pi_2 = \pi$
9. Let $V := (V \setminus \pi_1) \searrow \pi_1$
10. Let $E := \{(\pi_1, \pi_2, i) \in V \times V \times \mathbb{R} | \exists event_1 \in \pi_1, event_2 \in \pi_2: event_1 \in \pi_1, event_2 \in \pi_2\}$
11. End while
12. If (hybrid)
13. $(V, E) := kTail((V, E), 0, I)$
14. Output: $(V, E)$
FSM-inference representations

Formally, given two sequences of length $k$ $seq1 = (x_1^1, P_1^1)$ \ldots $(x_k^1, P_k^1)$ and $seq2 = (x_1^2, P_1^2)$ \ldots $(x_k^2, P_k^2)$, we say that

- $seq1$ $\equiv$ $seq2$ iff $\forall i = 1, \ldots, k \ x_i^1 = x_i^2$ and $P_i^1 \equiv P_i^2$
- $seq1 \sqsubseteq seq2$ iff $\forall i = 1, \ldots, k \ x_i^1 = x_i^2$ and $P_i^1 \Rightarrow P_i^2$

Given two $k$-futures $f1 = \{seq_1^1, \ldots, seq_{n1}^1\}$, with $seq_i^1 = (x_i^1, P_i^1)$ \ldots $(x_k^1, P_k^1)$ \ldots $(x_{n1}^1, P_{n1}^1)$ and $f2 = \{seq_1^2, \ldots, seq_{n2}^2\}$, with $seq_i^2 = (x_i^2, P_i^2)$ \ldots $(x_k^2, P_k^2)$ \ldots $(x_{n2}^2, P_{n2}^2)$, we say that

- $f1$ is eq
- $f1$ weakly subsumes $f2$ iff $n_1 = n_2$, $\forall j = 1, \ldots, n_1 \ x_j = 1, \ldots, n_1 \ s.t. \ seq_j^2 \subseteq seq_j^1$, and vice versa $\forall i = 1, \ldots, n_1 \ x_i = 1, \ldots, n_1 \ s.t. \ seq_i^1 \subseteq seq_i^2$
- $f1$ strongly subsumes $f2$ iff $\forall j = 1, \ldots, n_2 \exists i = 1, \ldots, n_1 \ s.t. \ seq_j^2 \subseteq seq_i^1$

Set theory

| 1 | Input: event log $L$ |
| 2 | let initialGraph = extract($L$) |
| 3 | let $I =$ mineInvariants(initialGraph) |
| 4 | let ($V, E)$ = partition(initialGraph) |
| 5 | while ($V, E$) does not satisfy invariants $I$ |
| 6 | // $p$: event -> boolean, $\pi$: partition that will be split |
| 7 | let $p$ |
| 8 | let $\pi$ |
| 9 | let $V$ |
| 10 | $E := \{(x_i^1, x_i^2, r) \in V \times V \times \mathbb{R} \mid \exists event_1 \in \pi_1, \exists event_2 \in \pi_2$ |
| 11 | : event_1, event_2 $\in$ initialGraph} |
| 12 | end while |
| 13 | if (hybrid) |
| 14 | ($V, E$) := kTail(($V, E)$, 0, I) |
| 15 | Output: ($V, E$) |
FSM-inference representations

Formally, given two sequences of length \(k\) and \(\text{seq1} = (x_1, P_{x_1}^1) \ldots (x_k, P_{x_k}^1)\) and \(\text{seq2} = (x_1', P_{x_1'}^2) \ldots (x_k', P_{x_k'}^2)\), we say that

- \(\text{seq1} = \text{seq2} \iff \forall i = 1 \ldots k \ x_i = x_i'\) and \(P_i = P_i'\)
- \(\text{seq1} \subseteq \text{seq2} \iff \forall i = 1 \ldots k \ x_i = x_i'\) and \(P_i \Rightarrow P_i'\)

Given two k-futures \(f1 = \{\text{seq1}, \ldots, \text{seq}_{k1}\}\) with \(\text{seq1} = (x_1, P_{x_1}^1) \ldots (x_k, P_{x_k}^1)\) and \(f2 = \{\text{seq2}, \ldots, \text{seq}_{k2}\}\) with \(\text{seq2} = (x_1', P_{x_1'}^2) \ldots (x_k', P_{x_k'}^2)\)

- \(f1\) is equivalent to \(\{1, \ldots, n_1\}\)
- \(f1\) weakly subsumes \(f2\) if \(n_1 = n_2, \ \forall j = 1 \ldots n_1 \exists i = 1 \ldots n_1 \ s.t. \ seq_{ki} \subseteq seq_{kj}\)
- \(f1\) strongly subsumes \(f2\) if \(\forall j = 1 \ldots n_2 \exists i = 1 \ldots n_1 \ s.t. \ seq_{ki} \subseteq seq_{kj}\)

Limitations:

- X Transparency
- X Extensibility
FSM-inference representations
Implementation of the kTails algorithm as defined in Hinsme & Faltings '92.

```java
public class kTails {
    public static Logger logger;
    static {
        logger = Logger.getLogger("kTails");
    }

    /**< Results kTails by applying kTails with the given k value to list k */
    public static PartitionGraph getPartitionGraph(List<List<Partition>> listk) {
        return pgraph;
    }

    /**< Finds all possible merges in graph. Requires making a new call to attemptMerges after every merge in case previously unmerge-able pairs become merge-able. */
    private static void attemptMerges(PartitionGraph pgraph, int k) {
        //Means finish of the merges that we want to perform.
        Set<PartitionMulti> merges = new LinkedHashSet<PartitionMulti>(pgraph);
    }
}
```

Formally, given two sequences of length \( k \) and \( seq_1 = (x_1^1, P_1^1) \ldots (x_k^1, P_k^1) \) and \( seq_2 = (x_1^2, P_1^2) \ldots (x_k^2, P_k^2) \), we say that:

- \( seq_1 \) \( \equiv \) \( seq_2 \) iff \( \forall i \ldots k \ x_i^1 = x_i^2 \) and \( P_i^1 \leftrightarrow P_i^2 \)
- \( seq_1 \subseteq seq_2 \) iff \( \forall i \ldots k \ x_i^1 = x_i^2 \) and \( P_i^1 \Rightarrow P_i^2 \)

Given two \( k \)-futures \( f_1 = \{ seq_1^1, \ldots seq_1^m \} \), with \( seq_1^i = (x_1^i, P_1^i) \ldots (x_k^i, P_k^i) \), and \( f_2 = \{ seq_2^1, \ldots seq_2^n \} \), with \( seq_2^j = (x_1^j, P_1^j) \ldots (x_k^j, P_k^j) \)

- \( f_1 \) is eq. \( 1 \ldots n_1 \)
- \( f_1 \) weakly subsumes \( f_2 \) iff \( n_1 = n_2, \forall j = 1 \ldots n_1, \exists i = 1 \ldots n_1 \ a.t.
\( seq_2^j \subseteq seq_1^i \) and vice versa \( \forall i = 1 \ldots n_1, \exists j = 1 \ldots n_1 \ a.t.
\( seq_1^i \subseteq seq_2^j \)
- \( f_1 \) strongly subsumes \( f_2 \) iff \( \forall j = 1 \ldots n_2, \exists i = 1 \ldots n_1 \ a.t.
\( seq_2^j \subseteq seq_1^i \)

---

**Why is this execution admitted?**

Observations

Limitations:

- **X** Transparency
- **X** Extensibility

---

Ivan Beschastnikh

University of Washington
FSM-inference representations

How to add features to an algorithm?

Observations

Limitations:

- Transparency
- Extensibility
1. Low-level specifications

2. Monolithic architecture

Observations

We limit the length of the sequences to be compared to a model of the observed behavior.

In the fourth step, we iteratively compare the k-futures of pairs of states to obtain a compact model.

By merging equivalent states, we can reduce the complexity of the model.

The procedure for merging equivalent states is as follows:

1. Low-level specifications

2. Monolithic architecture

Limitations:

- Transparency
- Extensibility

Transparency

Extensibility

...
InvariMint: modular and declarative

Prior work:

Observations → Model

InvariMint:

Observations → Property miner → Property Instances → Property composition → Model

Declarative specification of FSM-inference algorithm
**InvariMint: modular and declarative**

Prior work:

Observations → Model

InvariMint:

Observations → Property miner → Property Instances → Property composition → Model

**Declarative specification**

✓ Transparent
✓ Extensible
InvariMint: modular and declarative

Prior work:

Observations → Model

InvariMint:

Observations → Property miner → Property Instances → Model

Property types:

✓ Transparent
✓ Extensible
Outline

- FSM-inference algorithms not transparent/extensible
  - InvariMint -- modular and declarative FSM-inference

- Expressing algorithm using InvariMint
  - kTails -- a state-merging algorithm
  - Synoptic -- a state-splitting algorithm

- General specification formalism
- InvariMint evaluation
<table>
<thead>
<tr>
<th>src</th>
<th>dst</th>
<th>timestamp</th>
<th>type</th>
<th>message</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>6</td>
<td>ack</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>commit</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>7</td>
<td>ack</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>4</td>
<td>tx_commit</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>9</td>
<td>prepare</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>8</td>
<td>prepare</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>prepare</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>15</td>
<td>ack</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>11</td>
<td>commit</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>10</td>
<td>commit</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>12</td>
<td>tx_commit</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>5</td>
<td>tx_commit</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>4</td>
<td>tx_commit</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>prepare</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>22</td>
<td>commit</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>13</td>
<td>tx_commit</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>16</td>
<td>prepare</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>14</td>
<td>commit</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>19</td>
<td>commit</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>18</td>
<td>commit</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>20</td>
<td>tx_commit</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>21</td>
<td>tx_commit</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>21</td>
<td>tx_commit</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>17</td>
<td>commit</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>17</td>
<td>prepare</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>13</td>
<td>commit</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>12</td>
<td>tx_commit</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>11</td>
<td>tx_commit</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>10</td>
<td>commit</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>9</td>
<td>prepare</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>8</td>
<td>prepare</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>8</td>
<td>prepare</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>7</td>
<td>prepare</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>7</td>
<td>prepare</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>6</td>
<td>prepare</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>6</td>
<td>prepare</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>5</td>
<td>prepare</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>5</td>
<td>prepare</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>4</td>
<td>prepare</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>4</td>
<td>prepare</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>3</td>
<td>prepare</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
<td>prepare</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2</td>
<td>prepare</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>prepare</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>prepare</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>prepare</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>prepare</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>prepare</td>
<td></td>
</tr>
</tbody>
</table>

**kTails and Synoptic overview**

**Observations**

Ivan Beschastnikh

**Biermann et al.**

IEEE TC 2011

**Beschastnikh et al.**

ESEC/FSE 2011

**Model**

propose

abort

commit

tx-abort

tx-commit
kTails and Synoptic overview

Larger models: fewer behaviors

Smaller models: more behaviors

Observations

Smallest model
kTails($k$) and Synoptic overview

- Larger models: fewer behaviors
- Smaller models: more behaviors

Observations

Merging

Merge states followed by identical k-length execution sequences
**kTails** and Synoptic overview

Larger models: fewer behaviors

Smaller models: more behaviors

Merging

Observations

start

end

Larger models: fewer behaviors

Smaller models: more behaviors

Merging

Observations

start

end
**kTails(k) and Synoptic overview**

1. Choose any merge
2. Terminate when no more states can be merged

**Larger models:** fewer behaviors

**Smaller models:** more behaviors

**Merging**

**Observations**
**kTails(k) and Synoptic overview**

Larger models: fewer behaviors

Smaller models: more behaviors

**Split** states to eliminate executions that violate a set of mined observation invariants

Observations

Splitting

Smallest model

Start

End

Larger models: fewer behaviors

Smaller models: more behaviors

Split states to eliminate executions that violate a set of mined observation invariants
kTails(k) and Synoptic overview

Larger models: fewer behaviors

Smaller models: more behaviors

Observations

Splitting

Smallest model

Start

End

Observations
kTails(k) and Synoptic overview

Synoptic is non-deterministic:

Final model depends on splitting choices

1. Choose split to eliminate counter-examples to unsatisfied mined invariants:
   - x AlwaysFollowedBy y
   - x NeverFollowedBy y
   - x AlwaysPrecedes y

2. Terminate when model satisfies all mined invariants
Motivating questions

Larger models: fewer behaviors

Smaller models: more behaviors

Observations

kTails

Synoptic
Motivating questions

---

Larger models: fewer behaviors

Smaller models: more behaviors

---

How can we easily:

- ... get kTails to ignore certain \( k \)-length sequences?
- ... add the \( x \) AlwaysFollowedBy \( y \) invariant to kTails?
- ... make Synoptic deterministic?
- ... add a new kind of invariant to Synoptic?
- ... learn which properties kTails/Synoptic preserve?

---

We can answer all of these questions by representing these algorithms with InvariMint
InvariMint: modular and declarative

Observations → Property miner → Property Instances → Model

property types
Expressing $k$-Tails($k=1$)

kTails property types
Expressing $k$Tails ($k=1$)

A template to express merging of observation sequences that are identical in the first 1 event(s)

= 

A property type: “x can be immediately followed by one of $Y$”
Expressing \textit{kTails}(k=1)

Observations → Property miner → Property Instances → Model

“x can be immediately followed by one of Y”

<table>
<thead>
<tr>
<th>trace 1:</th>
<th>trace 2:</th>
</tr>
</thead>
<tbody>
<tr>
<td>login  check</td>
<td>login  check</td>
</tr>
<tr>
<td>check</td>
<td>compose</td>
</tr>
<tr>
<td>logout</td>
<td>send logout</td>
</tr>
</tbody>
</table>

Observations

\textit{kTails} output

Univeristy of Washington
Expressing $\text{kTails}(k=1)$

“$x$ can be immediately followed by one of $Y$”

<table>
<thead>
<tr>
<th>trace 1:</th>
<th>trace 2:</th>
</tr>
</thead>
<tbody>
<tr>
<td>login</td>
<td>login</td>
</tr>
<tr>
<td>check</td>
<td>check</td>
</tr>
<tr>
<td>check</td>
<td>check</td>
</tr>
<tr>
<td>logout</td>
<td>compose</td>
</tr>
<tr>
<td></td>
<td>send</td>
</tr>
<tr>
<td></td>
<td>logout</td>
</tr>
</tbody>
</table>

Observations:

$x=\text{login}$ property instance
Expressing $k$-Tails ($k=1$)

Observations -> Property miner -> Property Instances -> Model

"x can be immediately followed by one of Y"
Expressing $k$Tails ($k=1$)

Observations $\rightarrow$ Property miner $\rightarrow$ Property Instances $\rightarrow$ Model

"$x$ can be immediately followed by one of $Y$"

$=$

FSM template:

When to instantiate:

$$Eval(\log L, \langle x=a, Y=B \rangle) = \begin{cases} 
\text{true} : & \forall t \in L, \exists b \in B, \Diamond (a \rightarrow \Diamond b) \text{ in } t \land \\ 
\forall b \in B, \exists t \in L, \Diamond (a \rightarrow \Diamond b) \text{ in } t \\
\text{false} : & \text{otherwise}
\end{cases}$$
Expressing \( k \text{Tails}(k=1) \)

Observations → Property miner → Property Instances → Model

“\( x \) can be immediately followed by one of \( Y \)”

\[
\sum \setminus \set{x} \xrightarrow{x} s2 \xrightarrow{Y} s3 \xrightarrow{\Sigma \setminus Y} \sum \setminus \set{x}
\]

FSM template: $\xrightarrow{\Sigma \setminus \{x\}} s1 \xrightarrow{x} s2 \xrightarrow{Y} s3 \xrightarrow{\Sigma \setminus Y} \sum \setminus \{x\}$

Complete template

When to instantiate:

\[
Eval(\Log L, \langle x=a, Y=B \rangle) = \begin{cases} 
\text{true} : \forall t \in L, \exists b \in B, \Diamond (a \rightarrow \Box b) \text{ in } t \land \\
\forall b \in B, \exists t \in L, \Diamond (a \rightarrow \Box b) \text{ in } t \\
\text{false} : \text{otherwise}
\end{cases}
\]
Expressing $kTails(k=1)$

- This formulation is exact (identical models)
- Generalizes to arbitrary $k$ value

“$x$ can be immediately followed by one of $Y$”

$$\text{Eval}(\text{Log } L, (x=a, Y=B)) = \begin{cases} 
\text{true} : \forall t \in L, \exists b \in B, \Diamond (a \rightarrow \Box b) \text{ in } t \land \\
\forall b \in B, \exists t \in L, \Diamond (a \rightarrow \Box b) \text{ in } t \\
\text{false} : \text{otherwise}
\end{cases}$$
package synoptic.algorithms;

import java.util.ArrayList;
import java.util.Collections;
import java.util.LinkedHashSet;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.HashMap;
import java.util.LinkedHashMap;

import synoptic.algorithms.graphops.PartitionMultiMerge;
import synoptic.model.ChainsTraceGraph;
import synoptic.model.Partition;
import synoptic.model.PartitionGraph;
import synoptic.model.event.EventType;
import synoptic.model.interfaces.INode;
import synoptic.model.interfaces.ITransition;
import synoptic.util.InternalSynopticException;
import synoptic.util.NotImplementedSynopticException;

/**
 * Implements the K_Tails algorithm as defined in Biermann & Feldman '72.
 */
public class K_Tails {

    public static Logger logger;
    static {
        logger = Logger.getLogger("K_Tails");
    }

    /**
     * Constructs and returns a PartitionGraph generated by applying k_Tails with
     * the given k value to the given trace graph
     */
    public static PartitionGraph performK_Tails(ChainsTraceGraph g, int k) {
        PartitionGraph pGraph = new PartitionGraph(g, false, null);
        attemptMerge(pGraph, k);
        return pGraph;
    }

    /**
     * Finds all possible merges in pGraph. Requires making a new call to
     * attemptMerge after every merge in case previously un-merge-able pairs
     * become merge-able.
     */
    public static Set<List<PartitionMultiMerge>> possibleMerges(PartitionGraph pGraph) {
        List<Partition> partitions = pGraph.getPartitions();
        List<PartitionMultiMerge> merges = new ArrayList<PartitionMultiMerge>();
        for (int j = 0; j < partitions.size(); j++) {
            if (j == partitions.size() - 1) {
                continue;
            }
            Partition pi = partitions.get(j);
            Partition pj = partitions.get(j + 1);

            if (pj == null) {
                return null;
            }
            if (pi == null) {
                return null;
            }

            if (merge(pi, pj)) {
                merges.add(new PartitionMultiMerge(pi, pj, true, null));
            }
        }

        return merges;
    }

    private static boolean merge(Partition pi, Partition pj) {
        return true;
    }

    public static void attemptMerge(PartitionGraph pGraph, int k) {
        LinkedHashMap<String, LinkedHashMap<String, String>> mergedPartitions = new LinkedHashMap<String, LinkedHashMap<String, String>>();
        LinkedHashMap<String, LinkedHashMap<String, String>> allVisitedMatches = new LinkedHashMap<String, LinkedHashMap<String, String>>();

        for (Partition pi : pGraph.getPartitions()) {
            LinkedHashMap<String, LinkedHashMap<String, String>> allVisitedMatchesForPi = new LinkedHashMap<String, LinkedHashMap<String, String>>();
            for (Partition pj : pGraph.getPartitions()) {
                if (pj == null) {
                    continue;
                }
                if (pi == null) {
                    continue;
                }

                if (merge(pi, pj) && !allVisitedMatches.containsKey(pi.getName())) {
                    allVisitedMatches.put(pi.getName(), pj.getName());
                    allVisitedMatchesForPi.put(pi.getName(), pj.getName());
                }
            }
            mergedPartitions.put(pi.getName(), allVisitedMatchesForPi);
        }
    }

    private static boolean merge(Partition pi, Partition pj) {
        return true;
    }

    public static void main(String[] args) {
        // TODO: Implement k_Tails algorithm
    }
}
for (ITransition<NodeType> t1 : n1Trans) {
  NodeType c1 = t1.getTarget();
  // Skip c1 if it was visited by this method earlier.
  if (visitedN1Children.contains(c1)) {
    continue;
  }

  boolean kEqual = false;

  // Make sure to get transitions of the same relation.
  for (ITransition<NodeType> t2 : n2Trans) {
    .getTransitionsWithExactRelations(t1.getRelation()) {
      NodeType c2 = t2.getTarget();

      // Skip c2 if it was visited by this method earlier.
      if (visitedN2Children.contains(c2)) {
        continue;
      }

      // Skip c2 if its already been mapped to a c1 previously in the
      // outer loop.
      if (childKEquivMatches.contains(c2)) {
        continue;
      }

      if (kEqualsWithoutSubsumption(c1, c2, k - 1, allVisitedMatches)) {
        kEqual = true;
        childKEquivMatches.add(c2);
        break;
      }

    } // end of for
  } // end of for

  // Could not find any kEqual c2 to match with c1.
  if (!kEqual) {
    // Remove the record of visiting n1 and n2.
    allVisitedMatches.remove(n1);
    return false;
  }
}
return true;
}
Expressing \textbf{kTails}(k)

- This formulation is exact (identical models)
- Generalizes to arbitrary k value

“x can be immediately followed by one of Y”

How can we:

- ... get kTails to ignore certain k-length sequences?
- ... add the \textbf{x AlwaysFollowedBy y} invariant to kTails?
- ... make Synoptic deterministic?
- ... add a new kind of invariant to Synoptic?
- ... learn which properties kTails/Synoptic preserve?
Expressing \( kTails(k) \)

How can we:

... get \( kTails \) to ignore certain \( k \)-length sequences?
... add the \( x \) AlwaysFollowedBy \( y \) invariant to \( kTails \)?
... make Synoptic deterministic?
... add a new kind of invariant to Synoptic?
... learn which properties \( kTails/Synoptic \) preserve?
Expressing $\text{kTails}(k)$

“$x$ can be immediately followed by one of $Y$”

How can we:

... get kTails to ignore certain $k$-length sequences? ✓

... add the $x$ AlwaysFollowedBy $y$ invariant to kTails?

... make Synoptic deterministic?

... add a new kind of invariant to Synoptic?

... learn which properties kTails/Synoptic preserve?
Expressing \( k\text{Tails}(k) \)

```
\[
\begin{align*}
\text{Observations} & \rightarrow \text{Property miner} & \text{Property instances} & \rightarrow \text{Model}
\end{align*}
\]
```

“\( x \) can be immediately followed by one of \( Y \)”

How can we:

... get \( k\text{Tails} \) to ignore certain \( k \)-length sequences? ✓

... add the \( x \) AlwaysFollowedBy \( y \) invariant to \( k\text{Tails} \)?

... make \( \text{Synoptic} \) deterministic?

... add a new kind of invariant to \( \text{Synoptic} \)?

... learn which properties \( k\text{Tails} / \text{Synoptic} \) preserve?
Expressing Synoptic with InvariMint

Synoptic property types
Expressing **Synoptic** with **InvariMint**

Observations $\rightarrow$ Property miner $\rightarrow$ Property Instances $\rightarrow$ Model

$x$ AlwaysFollowedBy $y$

$x$ AlwaysPrecedes $y$

$x$ NeverFollowedBy $y$

**Synoptic invariants**
Expressing **Synoptic** with **InvariMint**

Observations → Property miner → Property Instances → Model

- x AlwaysFollowedBy y
- x AlwaysPrecedes y
- x NeverFollowedBy y

**Synoptic invariants**
Expressing **Synoptic** with **InvariMint**

Observations → Property miner → Property Instances → Model

- **x AlwaysFollowedBy y**

  ![Diagram 1](image1)

  - States: s1, s2
  - Transitions: x → s1, y → s2, x → s2
  - Invariants: $\Sigma \setminus \{x\}$, $\Sigma \setminus \{y\}$

- **x AlwaysPrecedes y**

  ![Diagram 2](image2)

  - States: s1, s3, s2
  - Transitions: x → s1, y → s1, s3 → x, s2 → s3, y → s2
  - Invariants: $\Sigma \setminus \{x, y\}$, $\Sigma \setminus \{x\}$, $\Sigma \setminus \{y\}$

- **x NeverFollowedBy y**

  ![Diagram 3](image3)

  - States: s1, s2
  - Transitions: x → s1, y → s1, s2 → s2, x → s2
  - Invariants: $\Sigma \setminus \{x, y\}$
Expressing Synoptic with InvariMint

Observations → Property miner → Property Instances → Model

Synoptic invariants

x AlwaysFollowedBy y

\( \Sigma \setminus \{x\} \quad \Sigma \setminus \{y\} \)

x AlwaysPrecedes y

\( \Sigma \setminus \{x, y\} \quad \Sigma \quad \Sigma \)

x NeverFollowedBy y

\( \Sigma \setminus \{x\} \quad \Sigma \setminus \{y\} \)
Expressing Synoptic with InvariMint

This formulation is approximate:

InvariMint model superset of Synoptic models

Synoptic invariants
How can we:

... get kTails to ignore certain k-length sequences? ✓

... add the $x \text{ AlwaysFollowedBy } y$ invariant to kTails?

... make Synoptic deterministic?

... add a new kind of invariant to Synoptic?

... learn which properties kTails/Synoptic preserve?
Expressing **Synoptic** with **InvariMint**

How can we:

- ... get $k$Tails to ignore certain $k$-length sequences? ✓
- ... add the $x$ AlwaysFollowedBy $y$ invariant to $k$Tails? ✓
- ... make Synoptic deterministic?
- ... add a new kind of invariant to Synoptic?
- ... learn which properties $k$Tails/Synoptic preserve?
How can we:

... get kTails to ignore certain k-length sequences? ✓
... add the x AlwaysFollowedBy y invariant to kTails? ✓
... make Synoptic deterministic?
... add a new kind of invariant to Synoptic?
... learn which properties kTails/Synoptic preserve?
Expressing Synoptic with InvariMint

Observations → Property miner → Property Instances → Model

x AlwaysFollowedBy y
x AlwaysPrecedes y

How can we:

... get kTails to ignore certain k-length sequences? ✓
... add the x AlwaysFollowedBy y invariant to kTails? ✓
... make Synoptic deterministic? ✓
... add a new kind of invariant to Synoptic?
... learn which properties kTails/Synoptic preserve?
Expressing Synoptic with InvariMint

How can we:

... get kTails to ignore certain k-length sequences? ✓

... add the x AlwaysFollowedBy y invariant to kTails? ✓

... make Synoptic deterministic? ✓

... add a new kind of invariant to Synoptic?

... learn which properties kTails/Synoptic preserve?
Expressing Synoptic with InvariMint

Observations → Property miner → Property Instances → Model

How can we:

... get kTails to ignore certain k-length sequences? ✓
... add the x AlwaysFollowedBy y invariant to kTails? ✓
... make Synoptic deterministic? ✓
... add a new kind of invariant to Synoptic? ✓
... learn which properties kTails/Synoptic preserve?
Expressing **Synoptic** with **InvariMint**

Observations → Property miner → Property Instances → Model

- x AlwaysFollowedBy y
- x AlwaysPrecedes y

**How can we:**

- ... get kTails to ignore certain k-length sequences? ✓
- ... add the x AlwaysFollowedBy y invariant to kTails? ✓
- ... make Synoptic deterministic? ✓
- ... add a new kind of invariant to Synoptic? ✓
- ... learn which properties kTails/Synoptic preserve?
Expressing **Synoptic** with **InvariMint**

How can we:

- ... get kTails to ignore certain k-length sequences? ✓
- ... add the x AlwaysFollowedBy y invariant to kTails? ✓
- ... make Synoptic deterministic? ✓
- ... add a new kind of invariant to Synoptic? ✓
- ... learn which properties kTails/Synoptic preserve? ✓
Expressing Synoptic with InvariMint

How can we:

... get kTails to ignore certain length sequences?
... add this AlwaysFollowedBy invariant to kTails?
... make Synoptic deterministic?
... add a new kind of invariant to Synoptic?
... learn which properties kTails/Synoptic preserve?

InvariMint advantages:

✓ Transparency
✓ Extensibility
Outline

• FSM-inference algorithms not transparent/extensible
  ▶ InvariMint -- modular and declarative FSM-inference

• Expressing algorithm using InvariMint
  ▶ kTails -- a state-merging algorithm
  ▶ Synoptic -- a state-splitting algorithm

• General specification formalism

• InvariMint evaluation
InvariMint specifications

Observations → Property miner → Property instances → Model

Property types
InvariMint specifications

Observations → Property miner → Property instances → Property composition → Model

Property types
Composition function
InvariMint specifications

Observations → Property miner → Property instances → Property composition → Model

Property type 1
Property type 2
...

Composition function
InvariMint specifications

Observations → Property miner → Property instances → Property composition → Model

- Property 1: Parameterized FSM Evaluation function
- Property 2: Parameterized FSM Evaluation function

**Parameterized FSM (PFSM) -- property template**
- An FSM with variable-labeled transitions

**Evaluation function**
- Controls when a PFSM is instantiated as an FSM
InvariMint specifications

**Observations** → **Property miner** → **Property instances** → **Property composition** → **Model**

**Property 1:**
- Parameterized FSM Evaluation function

**Property 2:**
- Parameterized FSM Evaluation function

- **Parameterized FSM (PFSM) -- property template**
  - An FSM with variable-labeled transitions
- **Evaluation function**
  - Controls when a PFSM is instantiated as an FSM
InvariMint specifications

Observations → Property miner → Property instances → Property composition → Model

FSMs

Property 1:
- Parameterized FSM Evaluation function
- Composition function

An equation over FSMs

Property 2:
- Parameterized FSM Evaluation function

- Parameterized FSM (PFSM) -- property template
  - An FSM with variable-labeled transitions
- Evaluation function
  - Controls when a PFSM is instantiated as an FSM
Outline

• FSM-inference algorithms not transparent/extensible
  ‣ InvariMint -- modular and declarative FSM-inference

• Expressing algorithm using InvariMint
  ‣ kTails -- a state-merging algorithm
  ‣ Synoptic -- a state-splitting algorithm

• General specification formalism

• InvariMint evaluation
InvariMint expressiveness

- FSM-inference algorithms from prior work
  - Expressed kTails and Synoptic
  - Useful for understanding both algorithms
  - Noticed overlap in their InvariMint specifications!

- What can and can’t we express with InvariMint?
  - Limited to FSM algorithms that consider temporality
  - Constrains inference to composition of properties
  - Benefits from a formal background
InvariMint performance

• FSM inference useful for large systems
  ▶ Compact representation of mass executions
  ▶ A node at Google generates a million messages/day

• Does transparency and extensibility impact performance?
  ▶ InvariMint versions of kTails and Synoptic over 100x faster than procedural variants

Xu et al. SLAML 2010
Performance: kTails

- InvariMint kTails much faster than procedural kTails
  - Efficient FSM composition (e.g., intersection)
  - The full log is never maintained in memory
Performance: Synoptic

- Procedural Synoptic much slower than InvariMint Synoptic
  - Synoptic’s modeling check invariants
  - Synoptic’s refinement deals with concrete event instances
Contributions

FSM-inference algorithms are not transparent/extensible

InvariMint

Open source  synoptic.googlecode.com
Contributions

FSM-inference algorithms are not transparent/extensible

InvariMint
Declarative specification of FSM-inference algorithms

- Provides **insight** into how an algorithms works
- A **common language** for inference algorithms
- Simplifies **extension** of existing algorithms
- **Faster** than procedural equivalents

Open source  synoptic.googlecode.com