CS 520
Theory and Practice of Software Engineering
Fall 2020

Model inference of processes

November 3, 2020

Thursday (November 5)

- Fourth in-class exercise
- On model inference of processes (today is a prelude with useful info)
- Form 3-, 4-, or 5- person teams
  - Use Moodle to self-select a team; open from today until Thursday at noon
  - In your group’s description, let us know if your group wants us to set up a Zoom breakout room for Thursday

Problem

- Missing or inaccurate system documentation makes it challenging to understand the intended system behaviors
- Complex logs of unintended system behaviors makes it difficult to debug them

Goal

- Take as input a set of observed traces (usually represented as sequences of events) of a given system
- Automatically produce an inferred model (often represented as an FSA) that must accept all of the observed traces
  - May also accept some unobserved traces
Model inference tool: Architecture [Synoptic]

Shopping cart: API (written in PHP)

- invalid-coupon
- valid-coupon
- reduce-price
- check-out
- get-credit-card

BisimH algorithm

1. Extract a trace graph from a given log using the regular expressions
2. Mine invariants from that trace graph
3. Create initial inferred model by partitioning the trace graph
4. While (current inferred model violates invariants)
   1. Generate counterexample path through the current inferred model illustrating a violation of a given invariant
   2. Refine current inferred model based on that counterexample path to produce next inferred model satisfying that invariant
5. Coarsen current inferred model to produce final inferred model

BisimH algorithm

1. Extract a trace graph from a given log using the regular expressions
2. Mine invariants from the trace graph
3. Create initial inferred model by partitioning trace graph
4. While (current inferred model violates invariants)
   1. Generate counterexample path through the current inferred model illustrating a violation of a given invariant
   2. Refine current inferred model based on that counterexample path to produce next inferred model satisfying that invariant
5. Coarsen current inferred model to produce final inferred model
1. Extract: Log and regular expressions

- **Log**, e.g.,

- Regular expression(s), e.g.,
  Line parsing: (?<ip>\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})/(?<TYPE>.+)\.php

- Special events for INITIAL and TERMINAL. Each event specified as a triple <trace ID, timestamp, event type>, e.g., <74.15.155.103, 06/Jan/2011:07:24:13, check-out>

1. Extract: Trace graph

- Each **event** specified as a triple <trace ID, timestamp, event type>
  - Also special events for INITIAL and TERMINAL

- Each **trace** is a linear graph where:
  - Each event corresponds to a vertex
  - The total ordering among the events (specified by their timestamps) corresponds to the edges among the nodes

- A **trace graph** is the union of the set of traces
BisimH algorithm

1. Extract a trace graph from a given log using the regular expressions.
2. Mine invariants from the trace graph.
3. Create initial inferred model by partitioning trace graph.
4. While (current inferred model violates invariants)
   1. Generate counterexample path through the current inferred model illustrating a violation of a given invariant.
   2. Refine current inferred model based on that counterexample path to produce next inferred model satisfying that invariant.
5. Coarsen current inferred model to produce final inferred model.

2. Mine invariants

• Capture temporal relationships between event types in a given trace graph:
  - a Always Followed by b (a → b) e.g., INITIAL → check-out
  - a Never Followed by b (a ≈ b) e.g., valid-coupon ≈ invalid-coupon
  - a Always Precedes b (a ← b) e.g., check-out ← get-credit-card

• Must be satisfied by all of the potential traces through that graph.

Shopping cart: Invariants

Use the Dwyer et al. property patterns.
BisimH algorithm

1. Extract a trace graph from a given log using the regular expressions
2. Mine invariants from the trace graph
3. Create initial inferred model by partitioning trace graph
4. While (current inferred model violates invariants)
   1. Generate counterexample path through the current inferred model illustrating a violation of a given invariant
   2. Refine current inferred model based on that counterexample path to produce next inferred model satisfying that invariant
5. Coarsen current inferred model to produce final inferred model
BisimH algorithm

1. Extract a trace graph from a given log using the regular expressions
2. Mine invariants from the trace graph
3. Create initial inferred model by partitioning trace graph
4. While (current inferred model violates invariants)
   1. Generate counterexample path through the current inferred model illustrating a violation of a given invariant
   2. Refine current inferred model based on that counterexample path to produce next inferred model satisfying that invariant
5. Coarsen current inferred model to produce final inferred model

Use counterexample guided abstraction refinement (CEGAR)

4. CEGAR: Generate counterexample path
   - Employ a model checker to determine if all potential paths through the current inferred model satisfy the invariants
     - If so, report satisfied
     - If not, report violated and a counterexample path illustrating a violation of one of the invariants

Shopping cart: Counterexample path

Violates invariant: valid-coupon \neq invalid-coupon

4. CEGAR: Refine
   1. Refine the current inferred model based on the violated invariant
      - Heuristically select a vertex relevant to that invariant
   2. Create the refined inferred model that satisfies that invariant
      - Split the selected vertex into multiple vertices
5. Coarsen

- Coarsen the refined inferred model
  - Search for vertices that didn't need to be split by the refine heuristic and merge them

- The coarsened inferred model satisfies all of the invariants. (It is the final inferred model.)
Shopping cart: Final model

Does the inferred model illustrate any unintended behavior (i.e. a bug)?

YES

Challenges for model inference

- How does the selection of the event sequence set affect the final model? affect performance?
- How does the parameterization of the model inference algorithm affect the final model?
- How do you compare/contrast (or diff) two inferred models?
- How to support other high-level language features such as concurrency, real-time constraints, …?

Learning from system traces

- Usage scenarios, e.g.,
  https://dl.acm.org/doi/pdf/10.1145/1656250.1656252
- Logs, e.g.,
  Synoptic, InvariMint
- Counterexample paths, e.g.,
  libalf: The automata learning framework
  (http://libalf.informatik.rwth-aachen.de)
  …
Homework 3:
Delta Debugging (optional)

- **Goal:** Given a program $P$ and a large test case $T$ on which $P$ fails for a particular reason, use an automated technique called delta debugging to reduce $T$ to a minimal test case which still causes $P$ to fail for the same reason.

- **Topics:** Delta debugging, bug report, minimal test case, bug fix

- **Due:** Thursday November 12, 2020, 9 PM EST

https://people.cs.umass.edu/~hconboy/class/2020Fall/CS520/hw3.pdf