Structural Sampling
for Statistical Software Testing

Michele Sebag
CNRS – Université Paris-Sud

Joint work: Nicolas Baskiotis, Marie-Claude Gaudel, Sandrine Gouraud

Entente Cordiale, 2007, May 21st
Contents

Autonomic Computing

Statistical Structural Software Testing

Machine Learning
  Discriminative vs Generative ML
  Prior knowledge and representation

S4T algorithm: Sampling for Statistical Structural Software Testing

Experimental Validation
"Considering current technologies, we expect that the total number of device administrators will exceed 220 millions by 2010."

-Gartner 6/2001
Autonomic Computing: Family of Applications

- Computer Science: ubiquitous complex systems
- Number of system administrators does not scale up
- Long term goal: Autonomic systems
- Starter: self-aware systems
- Behavioural modelling → Machine Learning
Some applications in Autonomic Computing

- Palatin-Wolf-Schuster
  Find misconfigured CPUs in a grid system
  KDD06

- Xiao et al.
  Active learning for game player modeling
  AAAI05

- Zheng et al.
  Use traces to identify bugs
  NIPS03-ICML06

- Baskiotis et al.
  Statistical Structural Software Testing
  IJCAI07
Software Testing

Algebraic approaches
   Model Checking

Statistical approaches
   For each test case, compare the program output/desired output
   Building set of test cases
      – Sampling the input variable domain
      – Sampling the program behaviours

miss exception branches
Statistical Structural Software Testing

Denise et al. ISSRE 2004

Program to be tested $\rightarrow$ Control Flow Graph

Finite State Automaton
set of nodes $\Sigma$, set of transitions $\mathcal{V} \subset \Sigma \times \Sigma$
Control Flow Graph → Execution paths

$N(v, \ell)$: number of paths of length $\ell$ starting at node $v$

\[
N(v, \ell + 1) = \sum_{w \text{ successor of } v} N(w, \ell)
\]

Uniform sampling of $T$-length paths

$s[0] = v_s$; start node

For $t = 1 \ldots T$

Let $v = s[t - 1]$

Select $w$ successor of $v$ with probability $\propto N(w, T - t)$

$s[t] = w$
From an execution path to an input case

Program

read (x, y) 1
if (x < 0) 2
    then x := −x; y := 1/y; 3
p := 1; 4
while (x > 0) 5
do p := p * y; x := x − 1; 6
print p; 7

Path

s = 1.2.4.5.7

Path → Constraint Satisfaction Problem

x ≥ 0 AND x ≤ 0

CSP → Test case

x = 0
Statistical Structural Software Testing, overview

Repeat

Program

Execution Path Sampler

Constraint Solver

UNFEASIBLE

FEASIBLE \equiv \text{Input case}

Until sufficient coverage

Set of test cases
From execution paths to test cases

Path $\rightarrow$ Constraint Solver $\rightarrow \begin{cases} \text{Feasible} & \rightarrow \text{Test case} \\ \text{Undecidable} \\ \text{Unfeasible} \end{cases}$

Limitation

- $\Pr(\text{path feasible } | \text{ real-world program}) \sim 10^{-10}, 10^{-15}$
- $\Rightarrow$ Decompose the program by hand !@#!
Discriminant Machine Learning

Learn the “Feasible Path” concept

- Input: a set of feasible/unfeasible paths
- Output: approximate the semantics of the program

```
Repeat
  Execution Path Sampler → Classifier → ML
  Constraint Solver
  UNFEASIBLE → FEASIBLE
  Input case
Until sufficient coverage
```

Set of test cases
Discriminant Machine Learning

Fails
- Very few positive examples
- Complex instance space

More feasible paths are required for discriminant learning

*But more feasible paths is all what SST needs*
Generative Machine Learning

Generate execution paths such that

- They are feasible
- They are new

The goal is

- Not reinforcement learning
- Not really active learning
- Generative learning

paths must be new
only feasible paths matter
not distribution estimation
Prior knowledge: What makes a path unfeasible?

**Violated dependencies**

if (x)
  then y := ...
  else z := ...

[...]
if (x)
  then u := ...
  else w := ...

$s = ..12457...$ is unfeasible.

**Domain dependent constraints**

If there are 17 or 19 uranium beams to be examined
the number of times in the loop is 17 or 19.

**Others**

Last time in the loop, execute closing instructions

**Non Markovian problems / Context sensitive grammars**
Parikh Map

\text{string} \rightarrow \text{count Ngrams}

Efficient for testing language equivalence

if two FSAs have $\sim$ Ngrams distribution,

$$\Pr(\text{FSAs are different}) < \varepsilon(N, \text{distance, samplesize, ..})$$

Efficient for learning context sensitive languages

\text{e.g. } a^n b^n \equiv |s|_a = |s|_b \text{ AND } |s|_{ba} = 0
Extended Parikh Map

- Count number of occurrences of symbols
- Record successor of $i$-th occurrence of symbols

$v \in \Sigma$
$(v, i) \in \Sigma \times \mathbb{N}$

\[
|v| : \Sigma^* \mapsto \mathbb{N} \\
\sigma_{v, i} : \Sigma^* \mapsto \Sigma
\]

$s|_v = \#v \text{ in } s$
$s|_{v, i} = \text{ successor of } i\text{-th occurrence of } v$

$s = vwvtxytx$  \rightarrow  

\begin{array}{c|c|c|c|c}
\hline
s & v & w & t & \cdots \\
\hline
2 & 1 & 2 & \cdots \\
\hline
w & t & v & x \\
\end{array}
Path Generation using Parikh Maps

Input $\mathcal{E} = \{(x_i, y_i), x_i \in \Sigma^*, y_i \in \{1, -1\}, i = 1..n\}$

Generation

$s[0] = v_s$
For $t = 1 \ldots T$
Let $v = s[t-1]$

Ideally:

$$s[t] = \arg\max_w \{Pr(s' \text{ feasible} \mid \text{prefix}(s') = sw)\}$$

not enough evidence
Path Generation using Parikh Maps, 2

Approximation

\[ |s|_v = i, \; |s|_w = j \]

\[ s[t] = \operatorname{argmax}_w \left\{ \Pr(s' \text{ feasible in } \mathcal{E} \mid |s'|_{v,i} = w, |s'|_w > j) \right\} \]
Path Generation using Parikh Maps, 2

Approximation

\[ s[t] = \arg\max_w \left\{ Pr(s' \text{ feasible in } E | s'_{v,i} = w, s'_{w} > j) \right\} \]

Fails: because of XORs
Mixing evidence from different subconcepts is misleading.

\[ s \text{ feasible iff } s_{v,1} = s_{v,3} \]
S4T: Overview

Prior knowledge

Feasible Path = $C_1 \lor \ldots \lor C_K$

Get rid of XORs

Init

for each $C_i$ represented in training set $\mathcal{E}^+$

identify $\hat{C}_i = \{s, s \in \mathcal{E}^+, s \prec C_i\}$

Generalize

for each $\hat{C}_i$, generate $s'$ “close” to $\hat{C}_i$

if $s'$ feasible, generalize $\hat{C}_i$
Init module

Feasible Path $= C_1 \lor \ldots \lor C_K$

Getting rid of XORS
By construction,

if $s_i$ and $s_j$ belong to $C_k$,

$\text{lgg}(s_i, s_j)$ is correct $\quad (\equiv \text{does not cover unfeasible paths})$
Init module: Getting rid of XORs, 2

\[ \hat{R}(s_i, s_j) \sim s_i \text{ and } s_j \text{ in same } C_k \]

for \( t = 1, \ell \)

Generate \( s \) in \( lgg(s_i, s_j) \)

Return \( False \) if \( s \) unfeasible

Return \( True \)

Complete, \( Pr(\neg \hat{R}(s_i, s_j)|R(s_i, s_j)) = 0 \)

Incorrect, \( p = Pr(\hat{R}(s_i, s_j)|\neg R(s_i, s_j)) \ll 1 \)

Find clusters after \( \hat{R} \)

For \( i = 1 \ldots n \)

Construct clique \( (s_i) \)
Init module: Construct clique($s_0$)

\[ \hat{C} = \{s_0\} \]
Define $H = \{s' \mid \forall s \in \hat{C}, R(s, s')\} \setminus \hat{C}$
Define $\text{degree}(s) = |\{s' \in H \mid R(s, s')\}|$
While ($H \neq \emptyset$)
Select $s_t = \arg\max_{s \in H}\{\text{degree}(s)\}$
\[ \hat{C} := \hat{C} \cup \{s_t\} \]
Init module: Construct clique($s_0$)

\[ \hat{C} = \{s_0\} \]

Define \[ H = \{ s' \mid \forall s \in \hat{C}, R(s, s') \} \setminus \hat{C} \]

Define \[ \text{degree}(s) = |\{ s' \in H \mid R(s, s') \}| \]

While \((H \neq \emptyset)\)

Select \(s_t = \arg\max_{s \in H} \{\text{degree}(s)\} \)

\[ \hat{C} := \hat{C} \cup \{s_t\} \]
Init module: Construct clique($s_0$)

\[ \hat{C} = \{s_0\} \]

Define \( H = \{s' \mid \forall s \in \hat{C}, R(s, s')\} \setminus \hat{C} \)

Define \( \text{degree}(s) = |\{s' \in H \mid R(s, s')\}| \)

While \( (H \neq \emptyset) \)

Select \( s_t = \arg\max_{s \in H} \{\text{degree}(s)\} \)

\[ \hat{C} := \hat{C} \cup \{s_t\} \]
Init module

Let $s_0$ in $C_0$; $C_0$ has $n_0$ representatives in $E$
$H$ contains all $n_0$ paths in $C_0$ plus
$r$ 'spurious' paths
with probability $p^r$

Probability $P_{err}$ of selecting a spurious path:

degree(path in $C_0) \geq n_0$
degree(spurious paths) $ler + B(n_0, p)$

\[ P_{Err} < r \times Pr(B(n_0, p) > n_0 - r) = r \times Pr(B(n_0, 1 - p) < r) \]

With

\[ Pr(B(n_0, 1 - p) < r) \leq \exp(-\frac{2}{n_0}(n_0(1 - p) - r)^2) \]

Probability of selecting a spurious path

\[ < \sum_{r=1}^{n-n_0} r \left(p \times e^{4(1-p)}\right)^r e^{-2r^2/n_0} < \frac{1-(pe^{4(1-p)-2/n})^{n-n_0+1}}{(1-pe^{4(1-p)-2/n})^2} \]
Generate module

Input $\hat{C}$

$\epsilon$-Greedy generation

Select

$$w = \arg\max \{ Pr(s' \text{ feasible in } \hat{C} \mid s'_{v,i} = w, |s'|_w > j) \}$$

When $\forall s'$ in $\hat{C}$ as above, select $w$ with probability $\epsilon$

Other variants
Multi-armed bandit
Near-miss based generalization
Select $s'$ infeasible nearest miss to $\hat{C}$
Generate $s''$ in $\text{lgg}(\hat{C}, s') \setminus \hat{C}$
Experimental Validation

Real world problem
36 nodes, 46 edges, length 240, \( \Pr(\text{feasible}) = 10^{-5} \)

Artificial problems
BNF grammar
Generate FSAs (\# nodes in 20,40; length in 120,250)
Construct target concept
easy problems: \( \Pr(\text{feasible}) \) in \( 10^{-3}, 10^{-2} \)
medium problems: \( \Pr(\text{feasible}) \) in \( 10^{-8}, 10^{-5} \)
hard problems: \( \Pr(\text{feasible}) \) in \( 10^{-15}, 10^{-10} \)

Performance
For each \( C \) represented in the training set
plot \((x, y)\): \( x \): initial representativity; \( y \) S4T representativity
Experimental Validation

**Final vs Initial coverage**

Final vs Initial coverage for $\epsilon = .1$, .5 and 1

Initial coverage $\rightarrow$ gain

$< 10\%$ 2 to 5 orders of magnitude

$< 30\%$ gain factor 3
Experimental Validation on FCT4: with and without Init module
Conclusion and Perspectives

Contributions
- Extended Parikh representation
- Getting rid of long range dependencies
- Exploration vs Exploitation

Next
- Finding $C_i$ non represented in the training set
- Coupling with software debugging

Zheng et al. 03-06