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

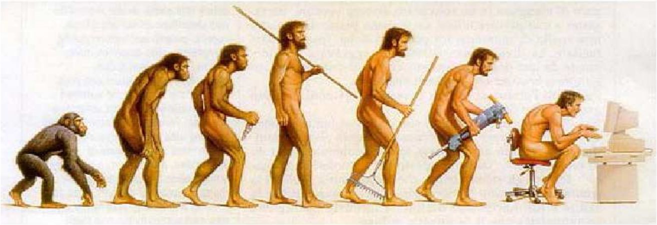
Statistical Structural Software Testing

Machine Learning
- Discriminative vs Generative ML
- Prior knowledge and representation

S4T algorithm: Sampling for Statistical Structural Software Testing

Experimental Validation

Evolution of Computing



"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 $\longrightarrow$ Machine Learning

Some applications in Autonomic Computing

- Palatin-Wolf-Schuster — KDD06
  Find misconfigured CPUs in a grid system
- Xiao et al. — AAAI05
  Active learning for game player modeling
- Zheng et al. — NIPS03-ICML06
  Use traces to identify bugs
- Baskiotis et al. — IJCAI07
  Statistical Structural Software Testing

Software Testing

Algebraic approaches

- Model Checking

Statistical approaches

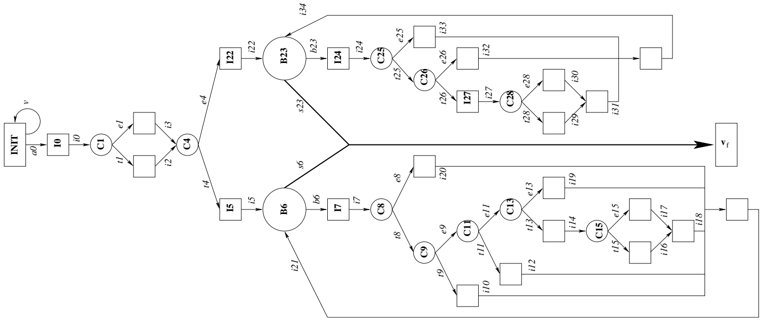
- For each test case, compare the program output/desired output
- Building set of test cases
  - Sampling the input variable domain

    miss exception branches
  - Sampling the program behaviours

Statistical Structural Software Testing

Denise et al. ISSRE 2004

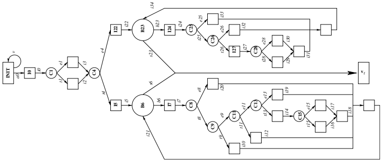
Program to be tested $\rightarrow$ Control Flow Graph



Finite State Automaton

set of nodes Σ, set of transitions $\mathcal{V} \subset \Sigma \times \Sigma$

Control Flow Graph $\longrightarrow$ Execution paths



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

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

Uniform sampling of T-length paths — Flajolet et al., TCS 94

$s[0] = v_s$; — 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

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

Path

$s = 1.2.4.5.7$

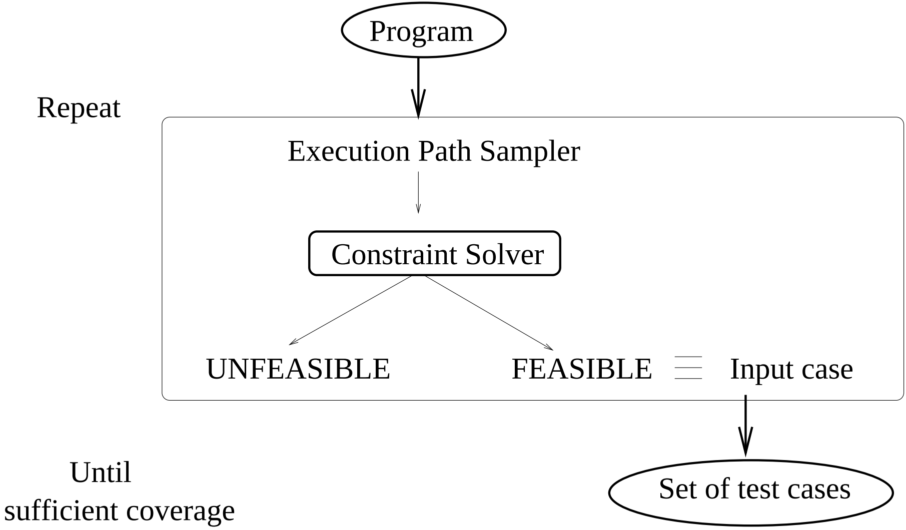
Path → Constraint Satisfaction Problem

$x \geq 0$ AND $x \leq 0$

CSP → Test case

$x = 0$

Statistical Structural Software Testing, overview



Statistical Structural Software Testing, overview 2

From execution paths to test cases

$$\text{Path} \longrightarrow \boxed{\text{Constraint Solver}} \longrightarrow \left\{ \begin{array}{lcl} \text{Feasible} & \longrightarrow & \text{Test case} \\ \text{Undecidable} & & \\ \text{Unfeasible} & & \end{array} \right.$$

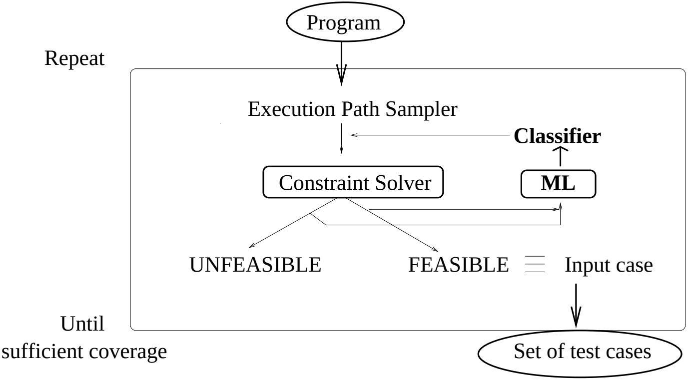
Limitation

- $Pr(\text{path feasible} \mid \text{real-world program}) \sim 10^{-10}, 10^{-15}$
- $\Rightarrow$ 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



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 — *paths must be new*
- Not really active learning — *only feasible paths matter*
- Generative learning — *not distribution estimation*

Prior knowledge: What makes a path unfeasible?

Violated dependencies	XOR
if (x)	1
then $y :=$ ...	2
else $z :=$ ...	3
[...]	4
if (x)	5
then $u :=$ ...	6
else $w :=$ ...	7

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

Domain dependent constraints — Loop

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

Representation

Parikh Map — Hopcroft-Ullman 79

string $\rightarrow$ count Ngrams

Efficient for testing language equivalence — Fischer et al. 04

if two FSAs have $\sim$ Ngrams distribution,

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

Efficient for learning context sensitive languages — Clark et al. 06

e.g. $a^n b^n \quad \equiv \quad |s|_a = |s|_b$ AND $|s|_{ba} = 0$

Extended Parikh Map

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

$$
\begin{array}{rll}
v \in \Sigma & |_v : \Sigma^* \mapsto \mathbb{N} & |s|_v = \#v \ in \ s \\
(v, i) \in \Sigma \times \mathbb{N} & |_{v,i} : \Sigma^* \mapsto \Sigma & |s|_{v,i} = \text{ successor of } \\
 & & \qquad \text{i-th occurrence of } v
\end{array}
$$

$$
s = vwvtxytx \quad \rightarrow \quad
\begin{array}{c|cccc|cccc}
 & |_v & |_w & |_t & \cdots & |_{v,1} & |_{v,2} & |_{w,1} & |_{t,1} \\
\hline
s & 2 & 1 & 2 & & w & t & v & x
\end{array}
$$

Path Generation using Parikh Maps

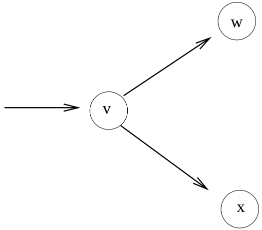
Input $\mathcal{E} = \{(x_i, y_i), x_i \in \Sigma^*, y_i \in \{1, -1\}, i = 1..n\}$

Generation

$s[0] = v_s$ start node

For $t = 1 \dots T$

Let $v = s[t-1]$



Ideally:

$$s[t] = argmax_w \{Pr(s' \text{ feasible } | prefix(s') = sw)\}$$

not enough evidence

Path Generation using Parikh Maps, 2

Approximation

$|s|_v = i, \; |s|_w = j$

$$s[t] = argmax_w\{Pr(s' \text{ feasible } in\ \mathcal{E}| \; |s'|_{v,i} = w, |s'|_w > j)\}$$

Path Generation using Parikh Maps, 2

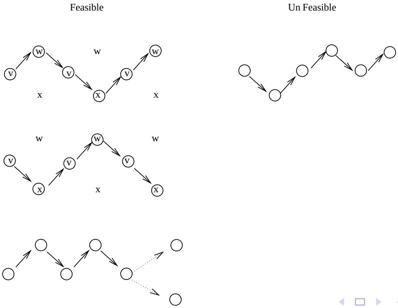
Approximation $\qquad |s|_v = i, \ |s|_w = j$

$$s[t] = argmax_w\{Pr(s' \text{ feasible } in\ \mathcal{E}|\ |s'|_{v,i} = w, |s'|_w > j)\}$$

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 — small disjuncts

$$\text{Feasible Path } = C_1 \vee \ldots \vee C_K$$

Init — Get rid of XORs

for each C_i represented in training set $\mathcal{E}^+$
identify $\widehat{C}_i = \{s, s \in \mathcal{E}^+, s \prec C_i\}$

Generalize

for each $\widehat{C}_i$,
generate s' "close" to $\widehat{C}_i$
if s' feasible, generalize $\widehat{C}_i$

Init module

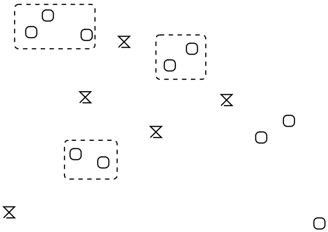
$$\text{Feasible Path } = C_1 \vee \ldots \vee C_K$$

Getting rid of XORS

By construction,

if s_i and s_j belong to C_k,

$lgg(s_i, s_j)$ is correct (≡ does not cover unfeasible paths)



Init module: Getting rid of XORs, 2

$\widehat{R}(s_i, s_j) \sim s_i$ and s_j 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, $p = Pr(\hat{R}(s_i, s_j)|\neg R(s_i, s_j)) << 1$

Find clusters after $\widehat{R}$

For $i = 1 \ldots n$

Construct clique (s_i)

Init module: Construct clique(s_0)

$\hat{C} = \{s_0\}$

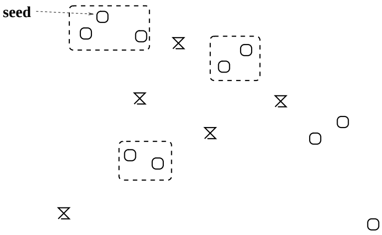
Define $\quad H = \{s' \;/\; \forall s \in \hat{C}, R(s, s')\} \setminus \hat{C}$

Define $\quad \text{degree}(s) = |\{s' \in H \;/\; R(s, s')\}|$

While $(H \neq \emptyset)$

$\quad$ Select $s_t = argmax_{s \in H}\{degree(s)\}$

$\quad \hat{C} := \hat{C} \bigcup \{s_t\}$



Init module: Construct clique(s_0)

$\hat{C} = \{s_0\}$

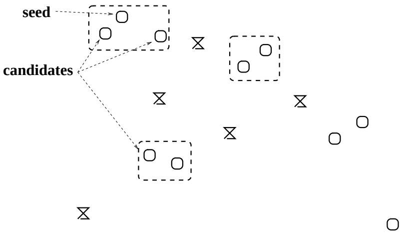
Define $\quad H = \{s' \;/\; \forall s \in \hat{C}, R(s, s')\} \setminus \hat{C}$

Define $\quad \mathrm{degree}(s) = |\{s' \in H \;/\; R(s, s')\}|$

While $(H \neq \emptyset)$

$\quad$ Select $s_t = argmax_{s \in H}\{degree(s)\}$

$\quad \hat{C} := \hat{C} \bigcup \{s_t\}$



Init module: Construct clique(s_0)

$\hat{C} = \{s_0\}$

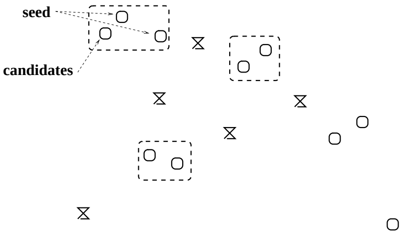
Define $\quad H = \{s' \ / \ \forall s \in \hat{C}, R(s, s')\} \setminus \hat{C}$

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

While $(H \neq \emptyset)$

Select $s_t = argmax_{s \in H}\{degree(s)\}$

$\hat{C} := \hat{C} \bigcup \{s_t\}$



Init module

Let s_0 in C_0; C_0 has n_0 representatives in $\mathcal{E}$
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) $ler + \mathcal{B}(n_0, p)$

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

With

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

Probability of selecting a spurious path

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

Generate module

Input $\widehat{C}$

ϵ-Greedy generation

Exploration vs Exploitation

Select

$$w = argmax\{Pr(s' \ feasible \ in \ \widehat{C}| \ |s'|_{v,i} = w, |s'|_w > j)\}$$

When $\nexists s'$ in $\widehat{C}$ as above, select w with probability ϵ

Other variants

- Multi-armed bandit
- Near-miss based generalization
  - Select s' infeasible nearest miss to $\widehat{C}$
  - Generate s'' in $lgg(\widehat{C}, s') \setminus \widehat{C}$

Experimental Validation

Real world problem

36 nodes, 46 edges, length 240, Pr(feasible) $= 10^{-5}$

Artificial problems BNF grammar

Generate FSAs (# nodes in 20,40; length in 120.250)

Construct target concept

- easy problems: Pr(feasible) in $10^{-3}, 10^{-2}$
- medium problems: Pr(feasible) in $10^{-8}, 10^{-5}$
- hard problems: Pr(feasible) in $10^{-15}, 10^{-10}$

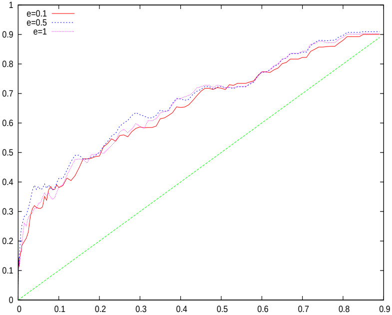
Performance

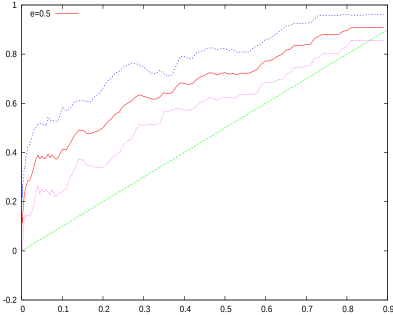
For each C represented in the training set

plot (x, y): x: initial representativity; y S4T representativity

Experimental Validation

Final vs Initial coverage





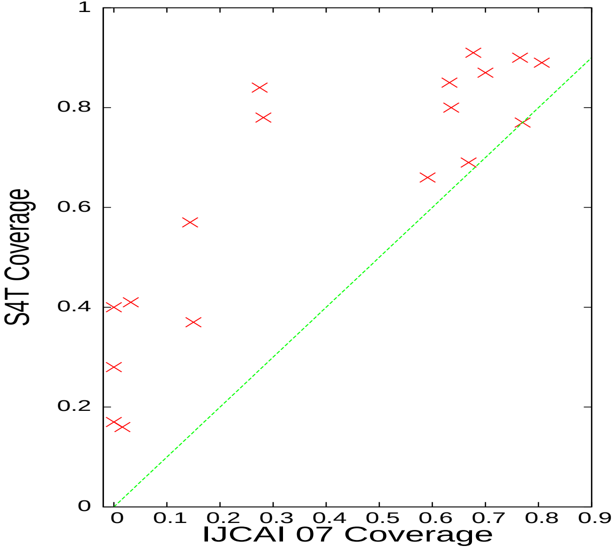
Final vs Initial coverage
for $\epsilon = .1, .5$ and 1

Final vs Initial coverage $\pm\sigma$
for $\epsilon = .5$

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