Drawing uniformly at random in dynamic sets of paths

Frédéric Voisin, Marie-Claude Gaudel

LRI, Univ Paris-Sud, CNRS, CentraleSupélec,
Université Paris-Saclay, France

Frederic.Voisin@lri.fr, marieclaude.gaudel@gmail.com

CLA 2019, Versailles, 1-2/07/2019

Motivations

Testing programs with a large number of execution paths:

- Randomised choice of execution paths in Control Flow Graphs (C.F.G.)
- Uniform coverage of paths up to a given length
- Exploration of large models or big amounts of data organised as graphs

Our issues:

- Eliminate from the drawing certain kinds of paths: infeasible paths, paths already drawn, etc.
- No prior knowledge about the kind of paths to be eliminated: checking feasibility of paths is delegated to an external procedure
- The set of paths to exclude increases with additional drawings
- Elimination is **"prefix based"**: all paths with a given prefix must no longer be drawn

This work might apply to other notions of "forbidden prefixes", besides infeasibility

Statistical Structural Testing

Structural Testing:

- Expressed as some coverage at run time of some elements of the C.F.G.
- Two-phase generation of tests:
  - Select a set of paths that covers a given criterion
  - For each path:
    Compute a formula ("path predicate") that characterizes any execution along that path
    Check with a SMT solver whether the formula is satisfiable and, if yes, derive input values.

    A "path predicate" is a conjunction $\Phi_1 \wedge \Phi_2 \cdots \wedge \Phi_q$

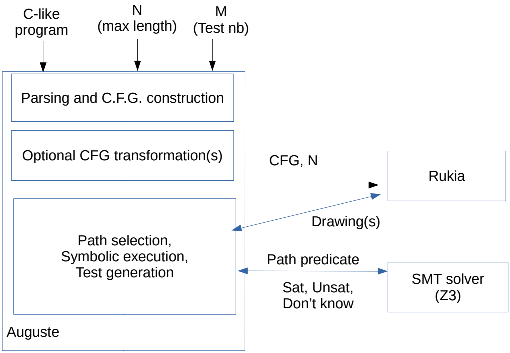
Randomising the selection of a set of paths:

- Uniform drawing among all paths of length up to a given bound
- Provides a natural alternative when exhaustiveness is out of reach
- Defines a way to assess the "quality" of a test set

Issues: Not all paths in C.F.G. are actual execution paths

- It is very common for a program to have infeasible paths; some programs have a huge ratio of infeasible paths
- SMT solvers have limitations (and satisfiability is undecidable for many logics) and high execution time
- Folks knowledge: the longer the path, the less chance to be feasible. We do not focus on producing "very long" paths.

The Auguste and Rukia tools



Rukia:

- C++ library on top of the Boost library,
- Implements a family of drawing algo. (recursive, Boltzmann generator, isotropic walks)
- Independent from application domains: uniform drawing in large graphs
  Available: http://rukia.lri.fr/en/index.html

Auguste:

- A family of prototypes for statistical structural testing
- Based on symbolic execution + SMT solvers for detecting infeasability
- Currently works on a subset of the C programming language.

Drawing with Rukia and the recursive method

- Specialisation of the classical recursive method for random generation of combinatorial structures [Wilf, Flajolet, and many others].
- Generate uniformly at random paths of length n from a graph $\mathcal{G}$ with root s_0 and final vertex s_f. We assume that s_0 has no in-going edge and s_f has no out-going edge.
- Statically compute a table $f(s,l)$ where s is a vertex and l a length
  where $f(s,l)$: nb of paths of length l from vertex s to s_f
  In particular, $f(s_0,n)$ is the number of paths of length n from s_0 to s_f.

The definition of f is:

$$f(s_i,j) = \sum_{s_i \to s_k \in \mathcal{G}} f(s_k, j-1), \qquad f(s,0) = 0 \text{ for } s \neq s_f, \qquad f(s_f,0) = 1 \tag{1}$$

Given $\mathcal{G}, n$ and f, Algorithm 1 draws a path p of length n from s_0 to s_f uniformly at random.

Algorithm 1 : drawing uniformly at random a path p of length n

$s = s_0;\ p = s_0;\ l = n;$
while $(l > 0)$ {
   draw s' among the successors s_k of s with probabilities $f(s_k, l-1)/f(s,l)$;
   $s = s';\ p = p.s';\ l = l-1;$
}

To draw paths of length $\leq n$ from s_0 to s_f, we add a fake edge from s_f to itself.

Contributions of this work

Initial situation:

- Rukia efficiently draws paths of hundreds of edges in graphs with more than 10^9 vertices.
- From our experiments, neither the size of the counting table, nor the time spent for drawing is currently a problem. Most time is spent at checking feasibility of paths.
- Infeasible paths are simply rejected and the drawing continues from the full collection.

Contributions:

- Given a set $\mathcal{F}$ of infeasible prefixes, discard for the subsequent drawings all paths with one of these prefixes
- Extend incrementally $\mathcal{F}$ according to new drawings
- Keep uniformity of the drawing among the remaining paths

Sometimes, redundant generation must be avoided. It is a special case of the problem above.

Why Focussing on "infeasible prefixes"

Infeasibility and Program Testing:

- Infeasibility of a path is detected when after a prefix, the conjunction of current path predicate with the condition of a branching statement gives a formula that is insatisfiable.

  $$\begin{array}{lcl} \text{Path predicate}(p.s) & = & \Phi_1 \wedge \Phi_2 \cdots \wedge \Phi_q \\ \text{Path predicate}(p.s.s') & = & \Phi_1 \wedge \Phi_2 \cdots \wedge \Phi_q \wedge \Phi_{q+1} \end{array}$$

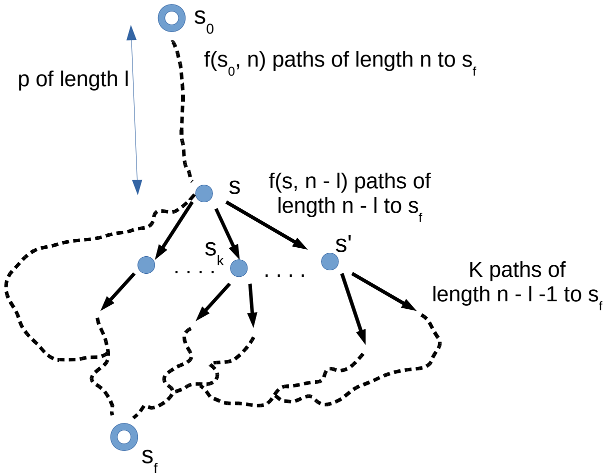
  where Φ_{q+1} is the condition for traversing the edge $s \rightarrow s'$ at that point in the program
- When a prefix is infeasible, so are all its possible extensions
- As a path predicate is built incrementally, checking always stops at the shortest infeasible prefix

Note that a prefix is never empty: if p is empty, then $s = s_0$.

Our notation of an "infeasible" prefix distinguishes the vertex whose addition makes the prefix infeasible: a prefix $p.s.s'$ is infeasible because of the addition of the edge $s \rightarrow s'$.

Suppose that a path of length n is drawn but detected as containing an infeasible prefix $p.s.s'$ (with no shorter such prefix). All paths with prefix $p.s.s'$ must be excluded from future drawings.

Let note l the length of $p.s$ and $K = f(s', n - l - 1)$.

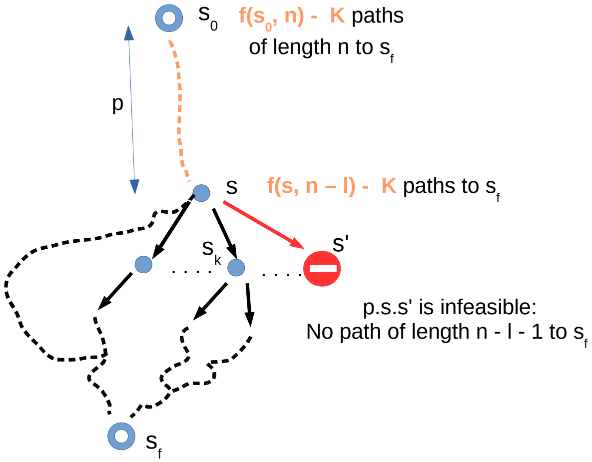


Counting and drawing from prefixes

Suppose that a path of length n is drawn but detected as containing an infeasible prefix $p.s.s'$ (with no shorter such prefix). All paths with prefix $p.s.s'$ must be excluded from future drawings.

Let note l the length of $p.s$ and $K = f(s', n - l - 1)$.

- Setting $f(s', n - l - 1)$ to 0 will prevent s' from being drawn as a successor for extending $p.s$.
- $f(s, n - l)$ has to be decremented by K, and the same must be done for all vertices along $p.s$, updating the counting table up to s_0 itself.

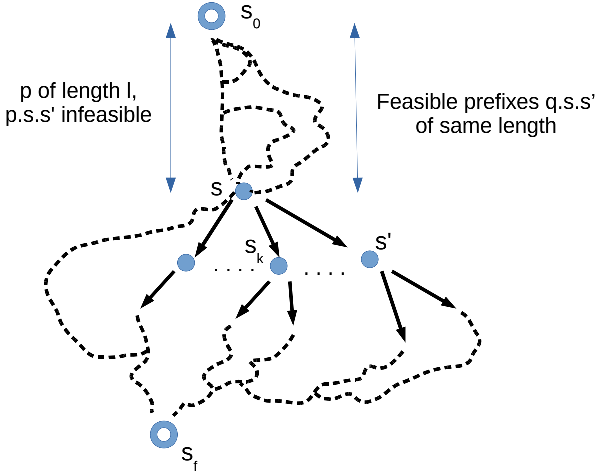


Counting and drawing from prefixes

Suppose that a path of length n is drawn but detected as containing an infeasible prefix $p.s.s'$ (with no shorter such prefix). All paths with prefix $p.s.s'$ must be excluded from future drawings.

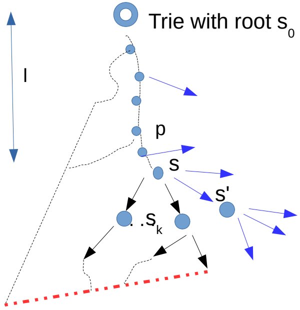
Let note l the length of $p.s$ and $K = f(s', n-l-1)$.

- Setting $f(s', n-l-1)$ to 0 will prevent s' from being drawn as a successor for extending $p.s$.
- $f(s, n-l)$ has to be decremented by K, and the same must be done for all vertices along $p.s$, updating the counting table up to s_0 itself.
- **But** for all feasible prefixes $q.s.s'$ of the same length, f would now give erroneous results. **Counting must be prefix dependent**.

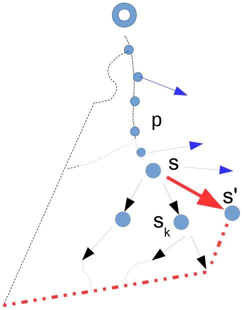


Main Ideas: Let $\mathcal{F}$ a set of infeasible prefixes

- Generalise f to a new counting table $f_{\mathcal{F}}$ with prefixes, not vertices, as first parameter
- Adapt Algo. 1 to use $f_{\mathcal{F}}$ when building prefix incrementally from s_0
- Build $f_{\mathcal{F}}$ lazily, defining entries only for prefixes yielding to infeasible paths
  - Use f for feasible prefixes: let $r.x \notin \mathcal{F}$, l its length, $f_{\mathcal{F}}(r.x, n-l) = f(x, n-l)$
  - Store the value of $f_{\mathcal{F}}$ for infeasibles prefixes in a **trie** $\mathcal{C}_{\mathcal{F}}$: the keys are prefixes and the value associated with a prefix r is $f_{\mathcal{F}}(r, n-|r|)$.



The blue part is not in the trie



The trie after handling p.s.s'

- The keys in $\mathcal{C}_{\mathcal{F}}$ are the infeasible prefixes and **all** their subprefixes;
- Elements from $\mathcal{F}$ appear only at leaves and have 0 associated with their key.

Drawing from $\mathcal{C}_{\mathcal{F}}$

Let $\mathcal{F}$ the set of infeasible prefixes, $\mathcal{C}_{\mathcal{F}}$ its trie, $f_{\mathcal{F}}$ and f the counting tables.

Algorithm 2 : drawing uniformly at random a path p of length n with $f_{\mathcal{F}}$

$let\ count(p.x, l) = if\ p.x\ \in\ \mathcal{C}_{\mathcal{F}}\ then\ return\ \mathcal{F}(p.x, l)\ else\ return\ f(x, l)$

$s = s_0;\ p = s_0;\ l = n;$
$while\ (l > 0)\ \{$
draw s' among the successors s_k of s with probabilities $count(p.s.s_k, l-1))/count(p.s, l)$
$s = s';\ p = p.s';\ l = l - 1;$
$\}$

Remark 1 : When starting Rukia, we make $\mathcal{C}_{\mathcal{F}}$ a trie reduced to root s_0 with initial value $f(s_0, n)$. Thus the drawing always starts within $\mathcal{C}_{\mathcal{F}}$.

Remark 2 : As long as one stays in $\mathcal{C}_{\mathcal{F}}$ the prefix currently built is feasible by construction. Rukia now returns not only a path but also the length of the stay within $\mathcal{C}_{\mathcal{F}}$: this can be used to avoid redundant feasibility checks.

Let $\mathcal{F}$ the set of forbidden prefixes, $\mathcal{C}_\mathcal{F}$ its trie, $f_\mathcal{F}$ and f the counting tables,
Let r a prefix of a path drawn with $\mathcal{C}_\mathcal{F}$ detected as infeasible, $\mathcal{F}' = \mathcal{F} \cup \{r\}$ and $\mathcal{C}_{\mathcal{F}'}$ its trie

Remark: The new drawing algorithm guarantees that, neither r nor one of its subprefix is in $\mathcal{F}$, otherwise r would not have been drawn.

Algorithm 3 : Updating $\mathcal{C}_\mathcal{F}$ to $\mathcal{C}_{\mathcal{F}'}$

let $K = f_\mathcal{F}(r, n - |r|)$;
Add a branch in $\mathcal{C}_\mathcal{F}$ labelled with the vertices from r using f for values of vertices not already in $\mathcal{C}_\mathcal{F}$;
For r and all its subprefixes, substract K from their value in $\mathcal{C}_\mathcal{F}$.

By construction r have associated value 0 in $\mathcal{C}_{\mathcal{F}'}$.

Updating the trie structure

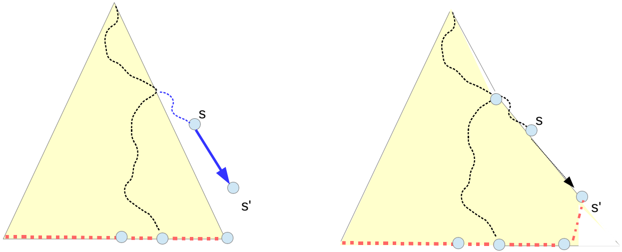
Let $\mathcal{F}$ the set of forbidden prefixes, $\mathcal{C}_{\mathcal{F}}$ its trie, $f_{\mathcal{F}}$ and f the counting tables,
Let r a prefix of a path drawn with $\mathcal{C}_{\mathcal{F}}$ detected as infeasible, $\mathcal{F}' = \mathcal{F} \cup \{r\}$ and $\mathcal{C}_{\mathcal{F}'}$ its trie

Remark: The new drawing algorithm guarantees that, neither r nor one of its subprefix is in $\mathcal{F}$, otherwise r would not have been drawn.

Algorithm 3 : Updating $\mathcal{C}_{\mathcal{F}}$ to $\mathcal{C}_{\mathcal{F}'}$

let $K = f_{\mathcal{F}}(r, n - |r|)$;
Add a branch in $\mathcal{C}_{\mathcal{F}}$ labelled with the vertices from r using f for values of vertices not already in $\mathcal{C}_{\mathcal{F}}$;
For r and all its subprefixes, substract K from their value in $\mathcal{C}_{\mathcal{F}}$.

Here, we illustrate that all subprefixes of $p.s.s'$ are now included in the trie and s' becomes a leaf.

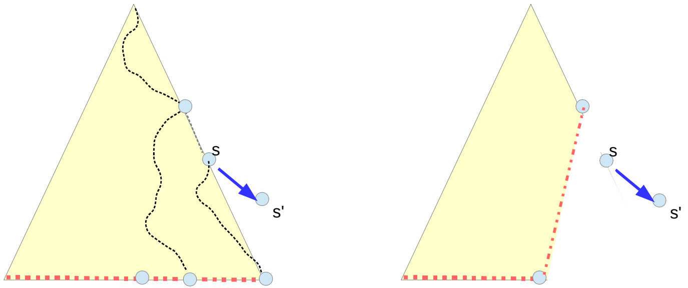


More on updating the trie structure

Implicit completion of $\mathcal{F}'$:
Propagating upwards the decrementation in $\mathcal{C}_{\mathcal{F}'}$ to vertices in r can set their value to 0: prefixes other than those in $\mathcal{F}'$ can have value 0 in $\mathcal{C}_{\mathcal{F}'}$.

Drawing such prefixes become impossible even if they are not in $\mathcal{F}'$. They are "dead ends" rather than "infeasible" but with a similar effect for drawing.

Pruning of $\mathcal{C}_{\mathcal{F}'}$:
Any subtree in $\mathcal{C}$ with a 0 value can be pruned, leaving the top-most vertice with 0 as leaf.



Remark: removing feasible paths in addition to infeasible ones leads to an implementation of an exhaustive search of feasible paths.

Experiments : Tcas

Context:

- One of the programs of the classical "Siemens benchmark" for testers
- Documentation mentions the presence of an infeasible path
- Several functions are a unique expression that we in-lined
- There is no loop but many lazy boolean operators, resulting in a complex flow of control
- CFG can be unfolded into a symbolic execution tree with 123 paths of length at most 47, all feasible

Experiments: Looking for feasible paths of length ≤ 50

- Drawing in the initial CFG
- Drawing in the Symbolic Execution Tree (optimal unfolding)
- Drawing in a CFG unfolded to a non optimal length (here 40 instead of 47)

Remarks:

- Here, exhaustiveness and avoiding duplicates are used to "stress" our method
- When testing an actual piece of code, usually you do not know any optimal value for unfolding, the number of feasible paths or whether these paths follow some special pattern.

Tcas: results

Current version: eliminates infeasibles *and* duplicates

	n	paths	drawn	infeasibles	alive	time	KP	size	SMT Saved	SMT Calls
tcas-opt	47	123	123	0	0	15.9	1	270	1378	901
tcas-CFG	47	179720	755	632	0	27,4	7562	1758	12043	1683
tcas-40	50	386	297	174	0	20,8	4	1127	4787	1240
tcas-40 (60)	50	386	158	98	151	15,5	4	1062	2258	958
CFG-40 (60)	50	181512	598	538	183	24,5	7562	1712	9097	1560

Previous version (drawing with replacement, no infeasible path elimination):

- tcas-opt: 123 over 123 paths; Drawings: 491 – 906;
- tcas-CFG: 123 over 179 720; Crash after 130 400 drawings with 62 feasibles found
- tcas-40 (60): 60 over 386; Drawings: 194;
- CFG-40 (60): 60 over 181 512; Drawings: 83 301;

Remark: Crash = "out of memory" on the Auguste side, not the Rukia side.

“Binary search”

Context:

- Inspired by an example from the ”Gallery” of Pathcrawler, a tool for concolic testing
- Initially: dichotomic search of an element in a sorted array
- As Auguste does not currently handle formulae for stating that an arrays is sorted, our program performs a kind of dichotomic walk in the array between two bounds that are input parameters.

Experiments: Looking for feasible paths of length ≤ 50

- Drawing in the initial CFG
- Drawing with a graph unfolded up to length 30 only
- Trying an exhaustive search of all feasible paths

"Binary search": results

Current version: eliminates infeasibles *and* duplicates

	n	paths	drawn	infeasibles	alive	time	KP	size	SMT Saved	SMT Calls
CFG-300	50	21247	791	491	5553	72,5	4607	6314	11674	4321
b30-300	50	8148	830	530	5171	78,3	31	6623	21612	4540
CFG-3000	50	21247	4883	2289	0	277,9	4607	10950	89484	14832
b30-3000	50	8148	4787	2293	0	291,7	31	10843	88443	14741

Remark: it finds the 2594 feasible paths and reports there are no more.
We had no information about the number of infeasible paths before the experiments.

Previous version (drawing with replacement, no infeasible path elimination):

- CFG-300: 300 over 21 247; Drawings: 2726 (2406 infeasible);
- b30-300: 300 over 8 148; Drawings: 944 (632 infeasibles);
- CFG-3000: 2594 over 21 247; Crash after 130 300 drawings (2590 found)
- b30-3000: 2594 over 8 148; Drawings: 83 369 (56 854 infeasible)

Conclusion and further works

Current state of achievement:

- Working prototype based on a modified version of Rukia
- We have presented only two experiments among all the ones we did.
- More experiments needed to assess scalability for program testing
- First experiments for program testing are rather promising:
  - Infeasibility is the best enemy of test generation, model checking and abstract interpretation techniques
  - Prefix-based elimination is a natural method
  - We observe a drastic improvement over the "drawing with rejection" method
- Efficiency depends on the killing ratio of prefixes: from a handful to thousands paths or more
- "Drawing without replacement" by itself is not competitive. It's an optional feature, at no cost.

Further works:

- Further reduce the size of graph and trie: Patricia trees or, when exporting a C.F.G., merge sequences of vertices with a unique successor in a single edge.
- Combining "prefix elimination" with Random Generation methods that would avoid producing patterns of infeasible paths: eg. embedded loops with some dependency on the number of iterations of the inner loop.
- Non uniform generation, for focusing on particular parts of a program

Let $\mathcal{F}$ the set of forbidden prefixes, $\mathcal{C}_\mathcal{F}$ its trie, $f_\mathcal{F}$ and f the counting tables,
Let r a prefix of a path drawn with $\mathcal{C}_\mathcal{F}$ detected as infeasible, $\mathcal{F}' = \mathcal{F} \cup \{r\}$ and $\mathcal{C}_{\mathcal{F}'}$ its trie

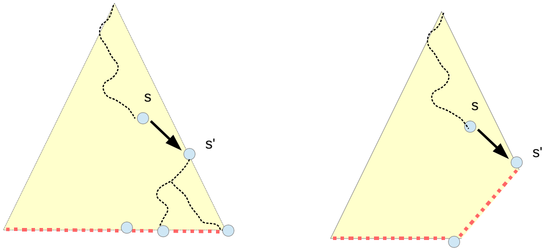
Remark: The new drawing algorithm guarantees that, neither r nor one of its subprefix is in $\mathcal{F}$, otherwise r would not have been drawn.

Algorithm 3 : Updating $\mathcal{C}_\mathcal{F}$ to $\mathcal{C}_{\mathcal{F}'}$

---

let $K = f_\mathcal{F}(r, n - |r|)$;
Add a branch in $\mathcal{C}_\mathcal{F}$ labelled with the vertices from r using f for values of vertices not already in $\mathcal{C}_\mathcal{F}$;
For r and all its subprefixes, substract K from their value in $\mathcal{C}_\mathcal{F}$.

Here, $p.s.s'$ has some infeasible extensions, but $p.s.s'$ itself is later deemed infeasible.
Note that this cannot happen if feasibility is detected incrementally.



Annex: Complexity results for Rukia

The following table is borrowed from Johan Oudinet's PhD thesis [2010]

As Rukia handles very large numbers, "binary complexity" is used.

n is the length of the path to draw and q is the number of vertices of the graph.

The use of *floating* indicates representing very large numbers by floating point numbers, not integer,

Method	Accuracy	Memory	PreProcessing	Drawing
recursive	exact	$\mathcal{O}(qn^2)$	$\mathcal{O}(qn^2)$	$\mathcal{O}(n^2)$
non tabular	exact	$\mathcal{O}(qn)$	$\mathcal{O}(q^2 + qn^2)$	$\mathcal{O}(qn^2)$
recursive	floating	$\mathcal{O}(qn \log n)$	$\mathcal{O}(qn \log n)$	$\mathcal{O}(q \log n)$
dichopile	floating	$\mathcal{O}(q^2 \log^2 n)$	$\mathcal{O}(1)$	$\mathcal{O}(qn \log^2 n)$
Boltzmann	floating	$\mathcal{O}(q)$	$\mathcal{O}(q^k \log^{k'} n)$	$\mathcal{O}(n^2)$