A Random Testing Approach using Pushdown Automata

A. Dreyfus, P.-C. Héam, O. Kouchnarenko, C. Masson

Département d'Informatique des Systèmes Complexes (DISC), Femto-ST UMR6174 CNRS

Outline

Random Exploration

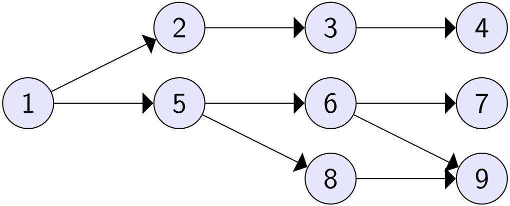
Random exploration optimizes the coverage...

- Robustness testing, Fuzz testing,
- Performance testing,
- Search based testing,
- Combination of random exploration with other techniques

Combination of graph-based testing and coverage criteria.

Alain Denise, Marie-Claude Gaudel, Sandrine-Dominique Gouraud, Richard Lassaigne, Johan Oudinet, and Sylvain Peyronnet. Coverage-biased random exploration of large models and application to testing. *STTT, 2012*.

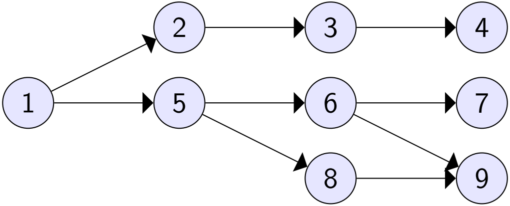
Testing on Graphs



All states,
all transitions,
all loops, etc.

Huge graphs lead to a huge number of tests.
Optimization of the number of tets can be hard (NP-complete).

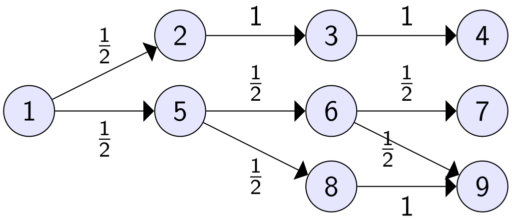
Testing on Graphs



All states,
all transitions,
all loops, etc.

Huge graphs lead to a huge number of tests.
Optimization of the number of tets can be hard (NP-complete).
Reduce the number of tests using a random approach.

Isotropic Random Walks



Paths of length 3.

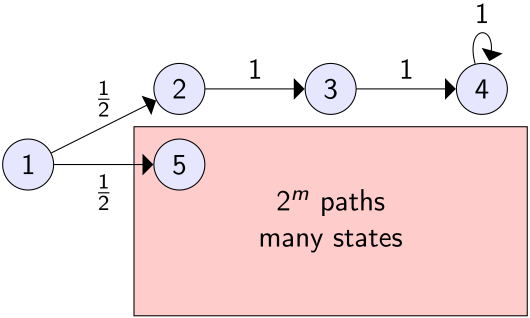
$1 \to 2 \to 3 \to 4$ probability $\frac{1}{2}$.

$1 \to 5 \to 6 \to 7$ probability $\frac{1}{8}$.

$1 \to 5 \to 6 \to 9$ probability $\frac{1}{8}$.

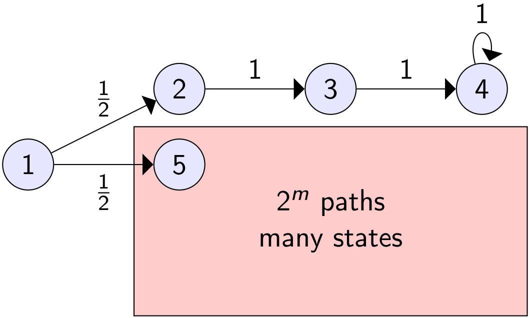
$1 \to 5 \to 8 \to 9$ probability $\frac{1}{4}$.

Isotropic Random Walks



$1 \to 2 \to 3 \to 4 \to \ldots \to 4$ probability $1/2$.
Other paths, probability $1/2^{m+1}$.

Isotropic Random Walks



$1 \to 2 \to 3 \to 4 \to \ldots \to 4$ probability 1/2.
Other paths, probability $1/2^{m+1}$.

Over representation of states 2, 3, 4 and related transitions

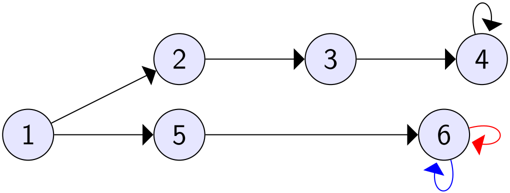
Local decisions don't provide a fair global coverage.

Uniform Generation of Paths

Each path of length n is picked up with the same probability.

Uniform Generation of Paths

Each path of length n is picked up with the same probability.
But some states/transitions can be visited by many paths of length n.



Prob. to cover 2: $\frac{1}{1+2^{n-2}}$

Prob. to cover 5: $\frac{2^{n-2}}{1+2^{n-2}}$

In this case the random walk approach provides a better state coverage.

Quality of the Random Process

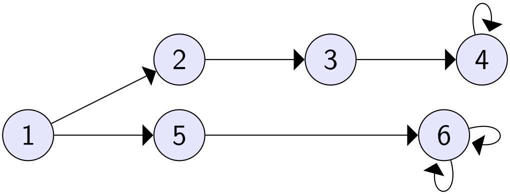
Probabilistic quality [T-FW89]

Given a random generation process, its quality $q_{C,N}$, relatively to a coverage criterion C, is the probability that all elements of C are covered when generating N test cases.

Quality of the Random Process

Probabilistic quality [T-FW89]

Given a random generation process, its quality $q_{C,N}$, relatively to a coverage criterion C, is the probability that all elements of C are covered when generating N test cases.



Unif. Gene. Paths.

$q_{\mathrm{states},N} = 1 - (1 - \frac{1}{1+2^{n-2}})^N$

Not good!

Random Walk

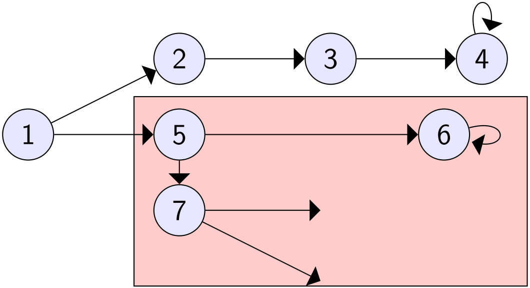
$q_{\mathrm{states},N} = 1 - \frac{1}{2^N}$

Very good!

Quality of the Random Process

Probabilistic quality [T-FW89]

Given a random generation process, its quality $q_{C,N}$, relatively to a coverage criterion C, is the probability that all elements of C are covered when generating N test cases.



Unif. Gene. Paths
(Very) Good!

Random Walk
Not good!

Random biased Generation

Algorithm [GDG+01]

Repeat N times:

1. Generate $c \in C$ randomly with probability π_c
2. Generate uniformly a path of length n visiting c.

[DGG+12]: how to compute π_c to optimize the quality.

Maximize $q_{\min}$ satisfying

$$\begin{cases} q_{\min} \leq \sum_{c' \in C} \mathrm{prob}(c, c')\pi_c & \text{for all } c \in C \\ \sum_{c \in C} \pi_c = 1 & \\ 0 \leq \pi_c \leq 1 & \text{for all } c \in C \end{cases}$$

prob(c,c'): probability that a path of length n visits both c and c'.

Random biased Generation

Algorithm [GDG+01]

Repeat N times:

1. Generate $c \in C$ randomly with probability π_c
2. Generate uniformly a path of length n visiting c.

[DGG+12]: how to compute π_c to optimize the quality.

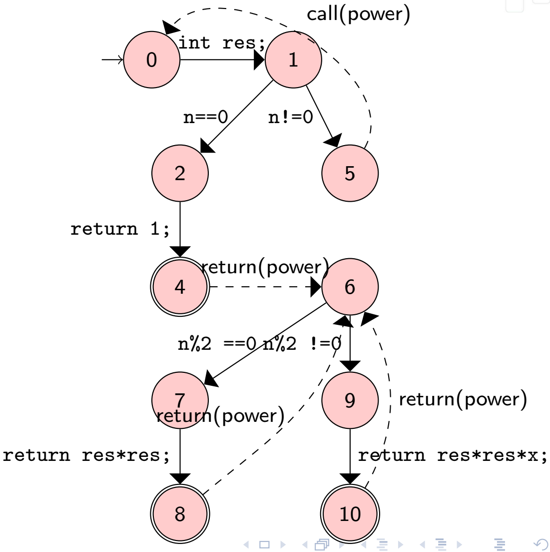
Requirements:

a. Computing the probability that a path visits both c and c'.
b. Solving a linear programming problem with $|C|$ variables.
c. Doing 2.

[DGG+12]: for graphs, for the state coverage criterion.

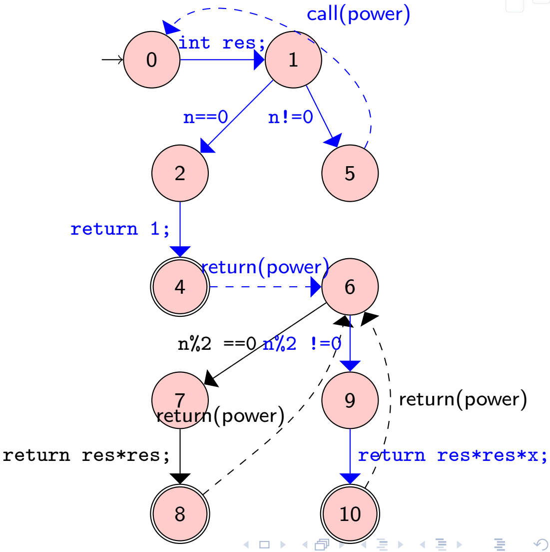
The Concretization Problem

```
int power(float x, int n){
    int res;
    if (n==0) {
        return 1;
    } else {
        res = power(x,n/2);
        if (n%2==0) {
            return res*res ;
        } else {
            return res*res*x
;
        }
    }
}
```



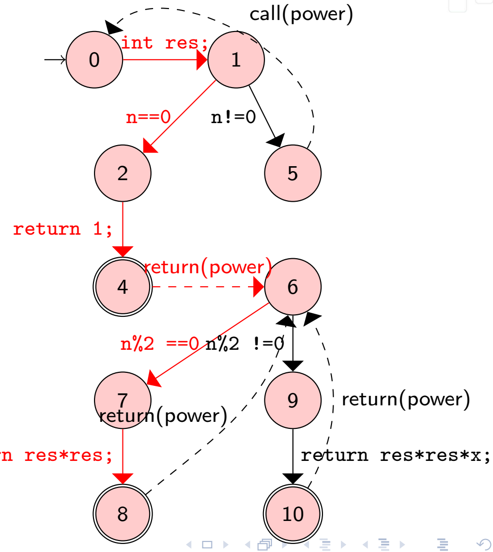
The Concretization Problem

```
int power(float x, int n){
    int res;
    if (n==0) {
        return 1;
    } else {
        res = power(x,n/2);
        if (n%2==0) {
            return res*res ;
        } else {
            return res*res*x
;
        }
    }
}
power(x,1)
```



The Concretization Problem

```
int power(float x, int n){
    int res;
    if (n==0) {
        return 1;
    } else {
        res = power(x,n/2);
        if (n%2==0) {
            return res*res ;
        } else {
            return res*res*x
;
        }
    }
}
```



This path cannot be concretized.

Motivations and Contributions

Main Issue

1. To avoid the concretization problem, use data flow graphs: a huge number of states (billions for simple programs).
2. Random biased approaches require quite small C (and therefore quite small graphs).

Goal

Develop similar techniques for graphs carrying information.

Our Work

Pushdown automata and state/transition coverage criteria.

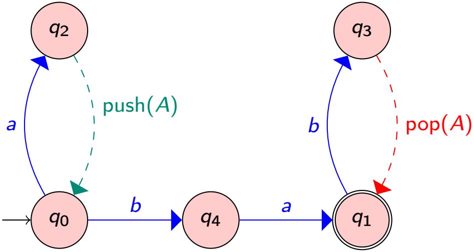
Outline

Modelling Systems using Pushdown Automata

Pushdown automata (PDA)

- Polynomial time decidability of emptiness, membership,
- Useful to model systems with (mutually) recursive functions,
- Useful to model parsing algorithms,
- Automatic tools to transform Java or C code into pushdown automata,
- Efficient model-checking tools.

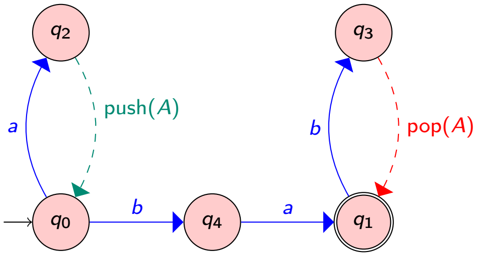
We consider Normalized Deterministic Pushdown Automata.



$\Gamma = \{A\}$ (stack)
$\Sigma = \{a, b\}$ (actions)
$Q = \{q_0, \dots, q_4\}$ (states)
$\Delta = \{(q_0, a, q_2), (q_2, \mathrm{push(A)}, q_0), (q_3, \mathrm{pop(A)}, q_1), \dots\}$ (transitions)

Each transition is labelled either by an action or a pop action or a push action.

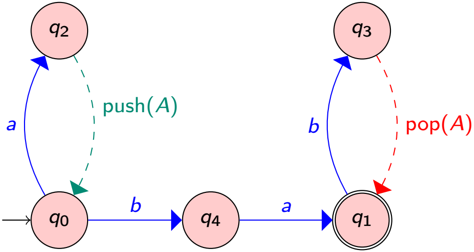
Configurations in PDA



$\Gamma = \{A\}$ (stack)
$\Sigma = \{a, b\}$ (actions)
$Q = \{q_0, \ldots, q_4\}$ (states)
$\Delta = \{(q_0, a, q_2), (q_2, \mathrm{push(A)}, q_0),$
$(q_3, \mathrm{pop(A)}, q_1), \ldots\}$ (transitions)

A *configuration* is a pair (q, w) where $w \in \Gamma^*$. The initial configuration is $(q_{\mathrm{init}}, \varepsilon)$. Final configurations are $(q_{\mathrm{final}}, \varepsilon)$.

Configurations in PDA

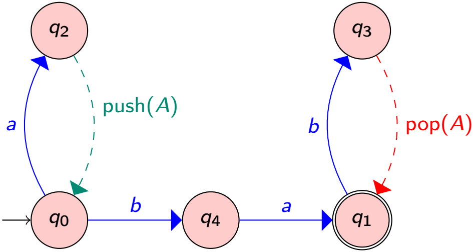


$\Gamma = \{A\}$ (stack)
$\Sigma = \{a, b\}$ (actions)
$Q = \{q_0, \ldots, q_4\}$ (states)
$\Delta = \{(q_0, a, q_2), (q_2, \mathrm{push(A)}, q_0),$
$(q_3, \mathrm{pop(A)}, q_1), \ldots\}$ (transitions)

A *configuration* is a pair (q, w) where $w \in \Gamma^*$. The initial configuration is $(q_{\mathrm{init}}, \varepsilon)$. Final configurations are $(q_{\mathrm{final}}, \varepsilon)$.

$(q, w) \rightarrow_a (q', w')$ if $(q, a, q') \in \Delta$ and $w = w'$

Configurations in PDA

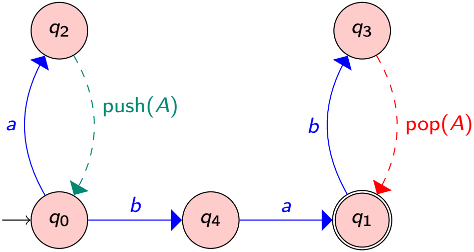


$\Gamma = \{A\}$ (stack)
$\Sigma = \{a, b\}$ (actions)
$Q = \{q_0, \ldots, q_4\}$ (states)
$\Delta = \{(q_0, a, q_2), (q_2, \mathrm{push(A)}, q_0), (q_3, \mathrm{pop(A)}, q_1), \ldots\}$ (transitions)

A *configuration* is a pair (q, w) where $w \in \Gamma^*$. The initial configuration is $(q_{\mathrm{init}}, \varepsilon)$. Final configurations are $(q_{\mathrm{final}}, \varepsilon)$.

$(q, w) \rightarrow_a (q', w')$ if $(q, a, q') \in \Delta$ and $w = w'$
$(q, w) \rightarrow_{\mathrm{pop(A)}} (q', w')$ if $(q, \mathrm{pop(A)}, q') \in \Delta$ and $w = w'A$

Configurations in PDA

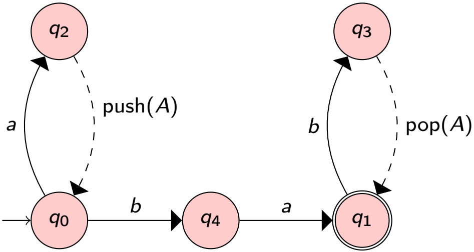


$\Gamma = \{A\}$ (stack)
$\Sigma = \{a, b\}$ (actions)
$Q = \{q_0, \dots, q_4\}$ (states)
$\Delta = \{(q_0, a, q_2), (q_2, \mathrm{push(A)}, q_0), (q_3, \mathrm{pop(A)}, q_1), \dots\}$ (transitions)

A *configuration* is a pair (q, w) where $w \in \Gamma^*$. The initial configuration is $(q_{\mathrm{init}}, \varepsilon)$. Final configurations are $(q_{\mathrm{final}}, \varepsilon)$.

$(q, w) \to_a (q', w')$ if $(q, a, q') \in \Delta$ and $w = w'$
$(q, w) \to_{\mathrm{pop(A)}} (q', w')$ if $(q, \mathrm{pop(A)}, q') \in \Delta$ and $w = w'A$
$(q, w) \to_{\mathrm{push(A)}} (q', w')$ if $(q, \mathrm{push(A)}, q') \in \Delta$ and $wA = w'$

Successful PDA-paths



$\Gamma = \{A\}$ (stack)
$\Sigma = \{a, b\}$ (actions)
$Q = \{q_0, \dots, q_4\}$ (states)
$\Delta = \{(q_0, a, q_2), (q_2, \mathrm{push(A)}, q_0), (q_3, \mathrm{pop(A)}, q_1), \dots\}$ (transitions)

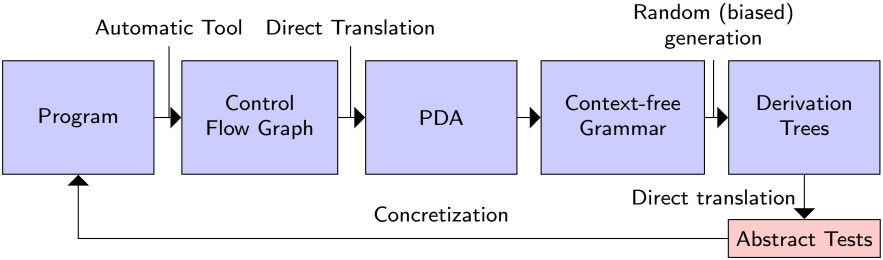
A *successful PDA-path* is a sequence of configurations linked by transitions, with initial and final conditions.

$$(q_0, \varepsilon) \rightarrow_a (q_2, \varepsilon) \rightarrow_{\mathrm{push(A)}} (q_0, A) \rightarrow_b (q_4, A) \rightarrow_a (q_1, A) \rightarrow_b (q_3, A) \rightarrow_{\mathrm{pop(A)}} (q_1, \varepsilon)$$

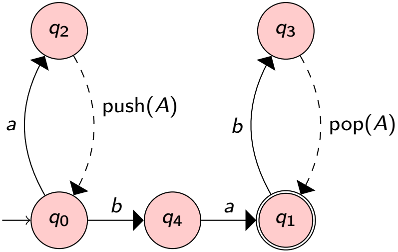
Given a path of length $4n + 2$ in the graph from q_0 to q_1, the probability that it corresponds to a successful PDA-path is $\frac{1}{2n+1}$.

Outline

Overview of the Approach (Doing 2.)

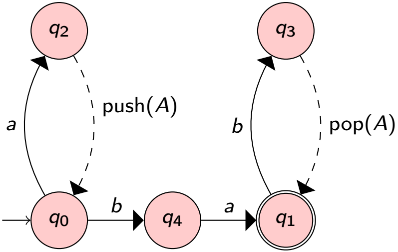


From Pushdown Automata to Grammars



From Pushdown Automata to Grammars

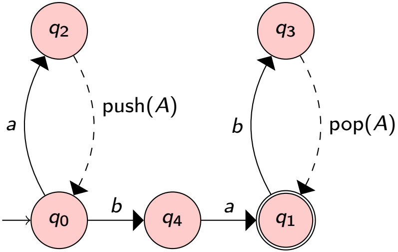
One can compute a grammar generating successful paths.



$$X_0 \to (q_0, a, q_2)X_2 \mid (q_0, b, q_4)(q_4, a, q_2)$$
$$X_2 \to (q_2, \mathrm{push(A)}, q_0)X_0S$$
$$S \to (q_1, b, q_3)(q_3, \mathrm{pop(A)}, q_1)$$

From Pushdown Automata to Grammars

One can compute a grammar generating successful paths.



$$X_0 \to (q_0, a, q_2)X_2 \mid (q_0, b, q_4)(q_4, a, q_2)$$
$$X_2 \to (q_2, \mathrm{push(A)}, q_0)X_0S$$
$$S \to (q_1, b, q_3)(q_3, \mathrm{pop(A)}, q_1)$$

Bijection Paths/D. Trees.

$$X_0 \to (q_0, a, q_2)X_2 \to (q_0, a, q_2)(q_2, \mathrm{push(A)}, q_0)X_0S$$
$$\to^* [(q_0, a, q_2)(q_2, \mathrm{push(A)}, q_0)]^2 X_0SS$$
$$\to [(q_0, a, q_2)(q_2, \mathrm{push(A)}, q_0)]^2(q_0, b, q_4)(q_4, a, q_2)SS$$
$$\to^* [(q_0, a, q_2)(q_2, \mathrm{push(A)}, q_0)]^2(q_0, b, q_4)(q_4, a, q_2)[(q_1, b, q_3)(q_3, \mathrm{pop(A)}, q_1)]^2$$

A Counting Approach [NW78][FZC94]

Example (a, b, c, d are terminal symbols):

$$E := EcE \mid EEd \mid a \mid b$$

The size of a derivation tree is its number of leaves.

- c_n number of trees of size n whose root rule is $E \to EcE$.
- d_n number of trees of size n whose root rule is $E \to EEd$.
- a_n number of trees of size n whose root rule is $E \to a$.
- b_n number of trees of size n whose root rule is $E \to b$.
- v_n number of trees of size n, $v_n = a_n + b_n + c_n + d_n$.

One has

- $v_1 = 2$, $v_2 = 0$ and $v_3 = 8$ $\qquad (a|b)c(a|b)$ and $(a|b)(a|b)d$.
- For example,

$$c_n = \sum_{k=1}^{n-1} v_k v_{n-k-1}.$$

- v_n, a_n, b_n, c_n and d_n can be computed recursively.

Random Generation of Trees [NW78][FZC94]

$E := EcE \mid EEd \mid a \mid b$ To generate derivation trees of size n.

Random Generation of Trees [NW78][FZC94]

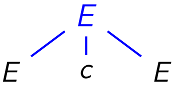
$E := EcE \mid EEd \mid a \mid b$ To generate derivation trees of size n.

1. Compute a_k, d_k, b_k, c_k, v_k for all $k \leq n$.

Random Generation of Trees [NW78][FZC94]

$E := EcE \mid EEd \mid a \mid b$ To generate derivation trees of size n.

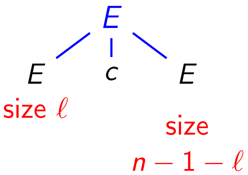
1. Compute a_k, d_k, b_k, c_k, v_k for all $k \leq n$.
2. Root rule is
   $E \rightarrow EcE$ with probability $\frac{c_n}{v_n}$,
   $E \rightarrow EEd$ with probability $\frac{d_n}{v_n}$,
   $E \rightarrow a$ with probability $\frac{a_n}{v_n}$ and
   $E \rightarrow b$ with probability $\frac{b_n}{v_n}$.



Random Generation of Trees [NW78][FZC94]

$E := EcE \mid EEd \mid a \mid b$ To generate derivation trees of size n.

1. Compute a_k, d_k, b_k, c_k, v_k for all $k \leq n$.
2. Root rule is
   $E \to EcE$ with probability $\frac{c_n}{v_n}$,
   $E \to EEd$ with probability $\frac{d_n}{v_n}$,
   $E \to a$ with probability $\frac{a_n}{v_n}$ and
   $E \to b$ with probability $\frac{b_n}{v_n}$.
3. Left child has size ℓ with probability $\frac{v_\ell v_{n-1-\ell}}{c_n}$.

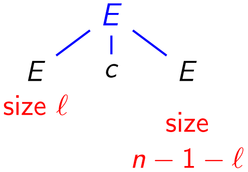


Note that v_n is the number of PDA-paths of length n.

Random Generation of Trees [NW78][FZC94]

$E := EcE \mid EEd \mid a \mid b$ To generate derivation trees of size n.

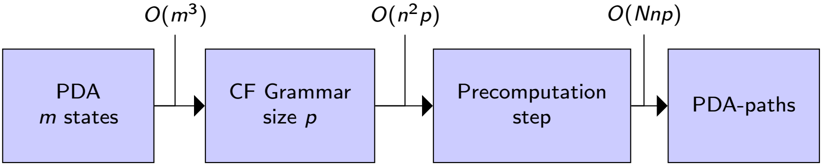
1. Compute a_k, d_k, b_k, c_k, v_k for all $k \leq n$.
2. Root rule is
   $E \to EcE$ with probability $\frac{c_n}{v_n}$,
   $E \to EEd$ with probability $\frac{d_n}{v_n}$,
   $E \to a$ with probability $\frac{a_n}{v_n}$ and
   $E \to b$ with probability $\frac{b_n}{v_n}$.
3. Left child has size ℓ with probability $\frac{v_\ell v_{n-1-\ell}}{c_n}$.
4. Recursive generation.



Note that v_n is the number of PDA-paths of length n.

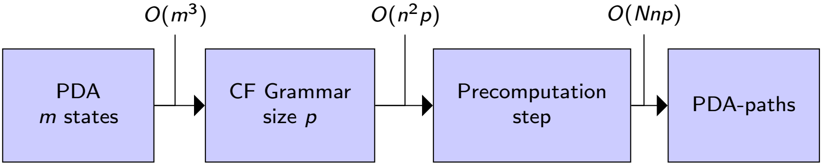
Complexity

Generation of N PDA-paths of length n.



Complexity

Generation of N PDA-paths of length n.



Cleaning step

Random biased Generation (Reminder)

Algorithm [GDG+01]

Repeat N times:

1. Generate $c \in C$ randomly with probability π_c
2. Generate uniformly a path of length n visiting c.

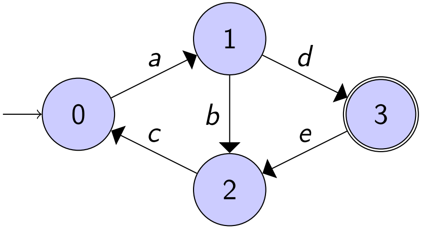
[DGG+12]: how to compute π_c to optimize the quality.

Requirements:

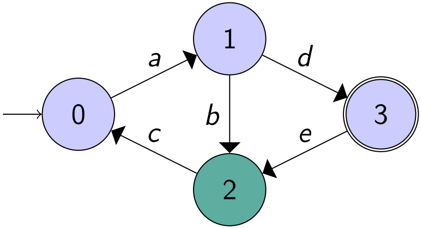
a. Computing the probability that a path visits both c and c'.
b. Solving a linear programming problem with $|C|$ variables.
c. Doing 2.

[DGG+12]: for graphs, for the state coverage criterion.

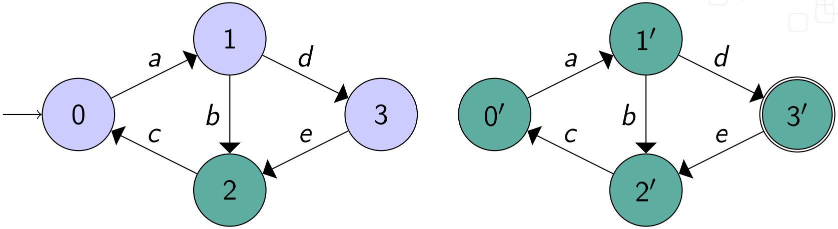
Random biased Generation for States



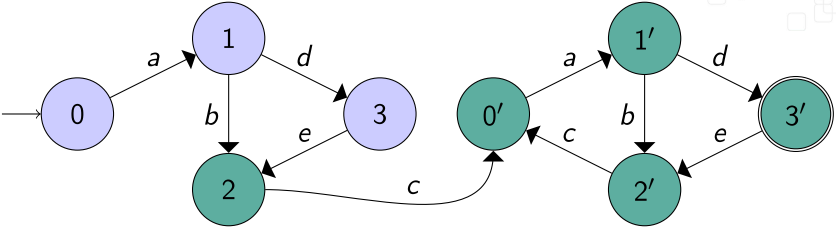
Random biased Generation for States



Random biased Generation for States

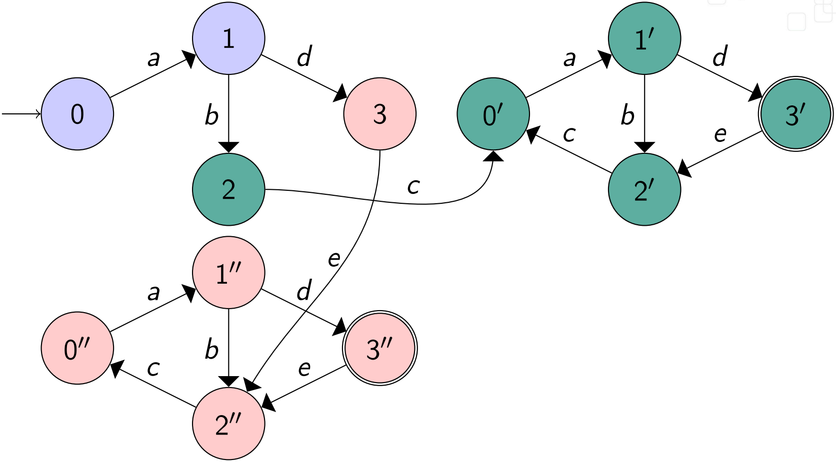


Random biased Generation for States



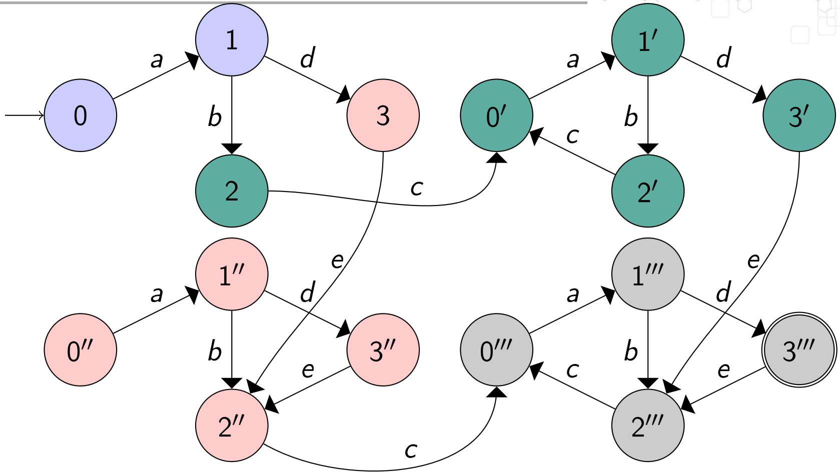
Allows the generation (and the enumeration) of paths of length n visiting 2.

Random biased Generation for States



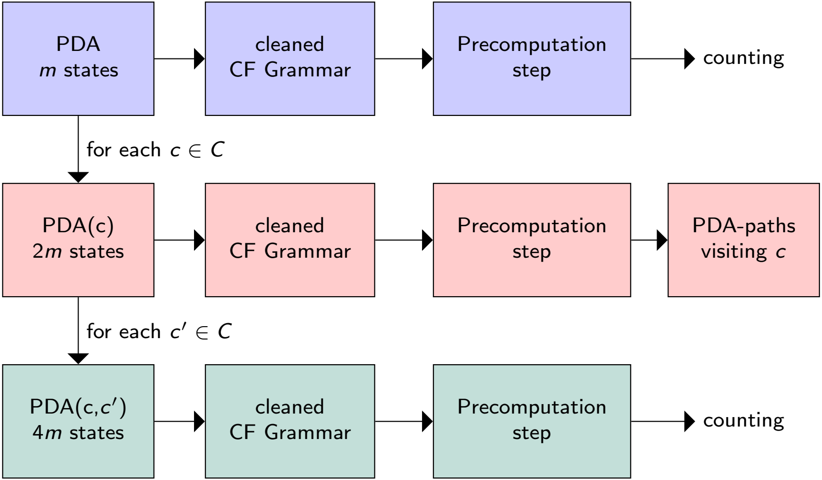
Allows the generation (and the enumeration) of paths of length n visiting 2 or 3.

Random biased Generation for States



Allows the generation (and the enumeration) of paths of length n visiting 2 and 3. Similar construction for transitions.

Overview



PDA
m states
cleaned
CF Grammar
Precomputation
step
counting
for each $c \in C$
PDA(c)
$2m$ states
cleaned
CF Grammar
Precomputation
step
PDA-paths
visiting c
for each $c' \in C$
PDA(c,c')
$4m$ states
cleaned
CF Grammar
Precomputation
step
counting

Using Coverage Criteria

Algorithm 1

Randomly and uniformly generates PDA-paths until C is covered.

Algorithm 2

Randomly and uniformly generates a PDA-path visiting a non already covered element until C is covered.

Algorithm 3

Randomly generates a PDA-paths with optimized probabilities element until C is covered.

Outline

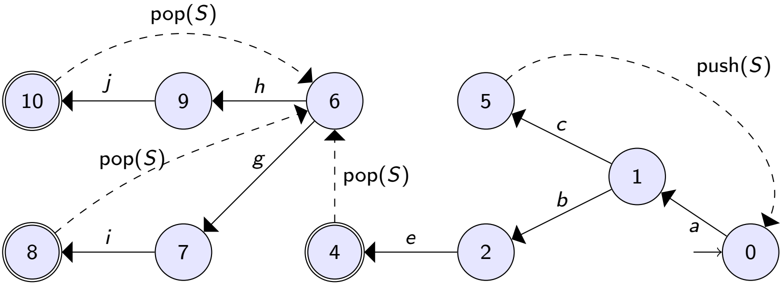
Examples, Shunting Yard Algorithm

Algorithm to translate an arithmetic expression into the equivalent expression in the Reverse Polish notation.

$$(3+2)*(5+4) \quad \rightarrow \quad 3\ 2\ +\ 5\ 4\ +\ *$$

The algorithm exploits a stack data structure.

Examples, the Power Function



Corresponding to the control flow graph of function computing x^k.

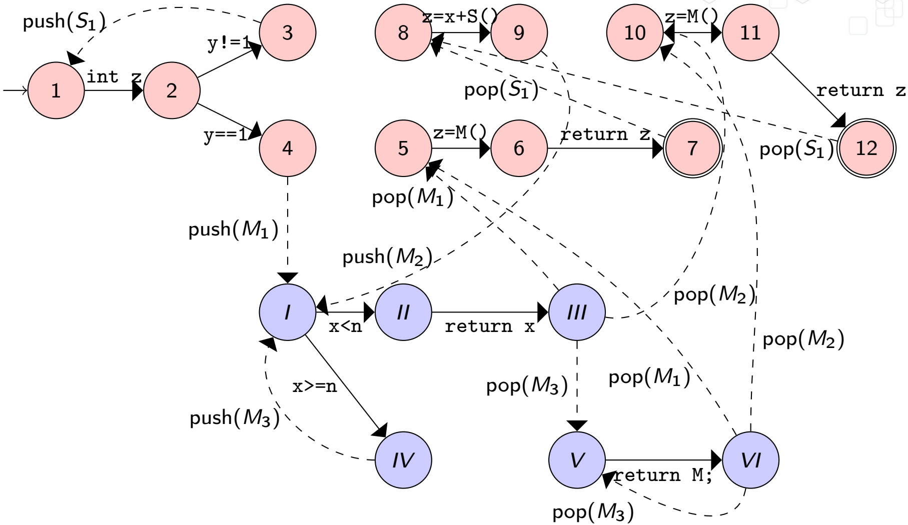
Examples, the Modulo Function

$M(x, n) = x$ modulo n and $S(x, y, n) = x * y$ modulo n

```
int S(int x, int y, int
n){
    int z;
    if (y == 1){
        z = M(x,n);
        return z;
    } else {
        z = x +
S(x,y-1,n);
        z = M(z,n);
        return z;
    }
}
```

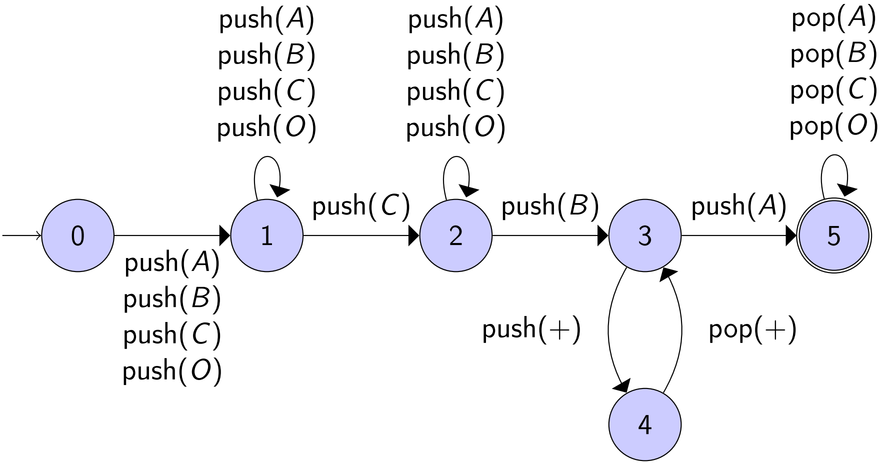
```
int M(int x, int n){
    if (x < 1){
        return x;
    } else {
        return M(x-n,n);
    }
}
```

Examples, the Modulo Function



Examples, XPATH Query

Modeling an XPATH query (from [C08]).



Examples, Sizes of the PDA

	states	transitions (1)	transitions (2)
Shunting Yard	23	36	102
Power	10	12	24
Modulo	18	24	90
XPATH	6	21	102

(1) Normalized transitions
(2) Considering all transitions

Sizes of the examples are comparable to those obtained by the automatic transformation into pushdown models of some industrial drivers [ST12].

NPDA vs. Finite Automata: $\mathcal{A}_{\text{modulo}}$

Path size	Our approach	Approach of [DGG+12]
from 1 to 7	no NPDA-trace	no DFA-trace
8	1-2-4-I-II-III-5-6-7 (10)	1-2-4-I-II-III-10-11-12 (6) 1-2-4-I-II-III-5-6-7 (4)
9	no NPDA-trace	no DFA-trace
10	no NPDA-trace	1-2-4-I-IV-I-II-III-5-6-7 (3) 1-2-4-I-IV-I-II-III-10-11-12 (2) 1-2-4-I-II-III-V-VI-10-11-12 (3) 1-2-4-I-II-III-V-VI-5-6-7 (2)
11	no NPDA-trace	A-2-3-1-2-4-I-II-III-10-11-12 (6) 1-2-31-2-4-I-II-III-5-6-7 (4)

NPDA vs. Finite Automata: $\mathcal{A}_{\text{modulo}}$

n	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
NPDA	1	0	0	0	1	0	0	1	1	0	0	1	1	0	0
DFA	6	4	10	6	18	10	34	18	64	34	114	64	200	114	356

n	27	28	29	30	31	32	33	34	35	36	37	80
NPDA	3	1	0	1	4	1	0	3	5	1	0	142
DFA	200	640	356	1152	640	2066	1152	3692	2066	6598	3692	$2.4\ 10^9$

For several lengths, as 21, 29, 37, there is no NPDA-traces. All generated DFA-traces would be not consistent.

Mutant based Evaluation

Using the PDA model of the Shunting Yard Algorithm and its implementation (from Wikipedia).

Uniform random generation of PDA-paths of length n.

Number of required test cases (20 experiments for each case).

	Killing mutants			Covering transitions			Cov. 79% of the code		
n	min	max	aver.	min	max	aver.	min	max	aver.
10	2	16	7.8	7	48	15.3	3	17	6.9
15	1	7	3.7	4	19	8.8	2	7	4.5
20	2	11	3.2	3	10	5.4	2	6	2.9

10 different mutants used: 4 Inc./Dec., 1 Arr. Ref. Replacement, 1 Switch Statement Mut., 4 Log. Neg.

Uniform Random Generation

	Grammar Size	Grammar Gen. Time(s)	Cleaned Grammar	Grammar Cleaning Time(s)	Precomp. Time(s)	100 Paths Gen. Time (s)
$\mathcal{A}_{\mathrm{xpath}}$	217+ 3463	0.03	27+87	0.38	(10) 0.07 (20) 0.22 (30) 0.57 (100) 17.21 (200) 141	(10) 0.09 (20) 0.17 (30) 0.63 (100) 1.1 (200) 2.83
$\mathcal{A}_{\mathrm{power}}$	201+369	0.02	27+31	0.05	(100) 0.98 (200) 5.63 (500) 78.96	(100) 0.41 (196) 0.94 (496) 3.94
$\mathcal{A}_{\mathrm{modulo}}$	1621+ 7572	0.07	45+50	6.07	(10) 0.03 (20) 0.08 (50) 0.41 (100) 2.29 (200) 14.86	(8) 0.07 (21) 0.14 (49) 0.32 (99) 0.28 (199) 1.4
$\mathcal{A}_{\mathrm{SY}}$	1937+ 16735	0.13	143+ 261	16.61	(10) 0.16 (20) 0.58 (30) 1.46 (100) 39.21 (200) 326.28	(10) 0.13 (20) 0.24 (30) 0.34 (100) 1.14 (200) 2.35

Using Coverage Criteria

Algorithm 1

Randomly and uniformly generates PDA-paths until C is covered.

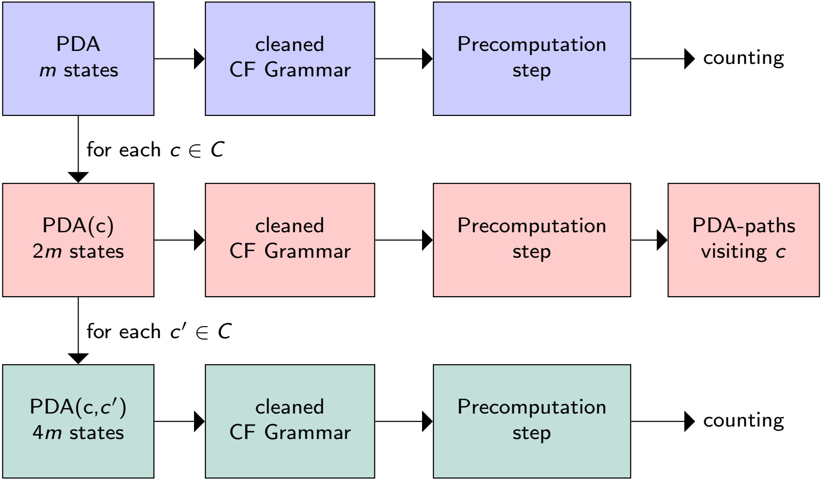
Algorithm 2

Randomly and uniformly generates a PDA-path visiting a non already covered element until C is covered.

Algorithm 3

Randomly generates a PDA-paths with optimized probabilities element until C is covered.

Overview


PDA
m states
cleaned
CF Grammar
Precomputation
step
counting
for each $c \in C$
PDA(c)
$2m$ states
cleaned
CF Grammar
Precomputation
step
PDA-paths
visiting c
for each $c' \in C$
PDA(c,c')
$4m$ states
cleaned
CF Grammar
Precomputation
step
counting

Computation time for Algorithm 2/3

From cleaned grammars: Computation time for
1) generating paths of length 60 visiting a given state/transition, and
2) computing their probabilities.

	$\mathcal{A}_{\mathrm{xpath}}$	$\mathcal{A}_{\mathrm{power}}$	$\mathcal{A}_{\mathrm{modulo}}$	$\mathcal{A}_{\mathrm{SY}}$
total time (state)	38s	5.25s	22min	136min
av. grammar size	31.3+98.8	32.2+36.8	65.2+72.9	342.7+692.9
total time (transition)	91s	6.3s	33min	219min
av. grammar sizes	30.5+94.8	30.7+34.3	66.2+73.7	273.3+572.6

Computation time for Algorithm 3

Computation time of the linear programming systems.

	$\mathcal{A}_{\text{xpath}}$	$\mathcal{A}_{\text{power}}$	$\mathcal{A}_{\text{modulo}}$	$\mathcal{A}_{\text{SY}}$
av. time (one pair of states)	13.4s	0.9s	408s	110min
total (state)	7min19s	79s	34h12min	847h (*)
total (transition)	25min	81s	60h39min	1200h (*)
total (simplified, state)	1min	insign.	9h (*)	120h (*)
total (simplified, transition)	8min (*)	insign.	18h (*)	350h (*)

(*) estimated time.

Simplifications based on symmetry/replication.

Comparing Algorithms for $\mathcal{A}_{\mathrm{xpath}}$

Average number of paths for several lengths, to cover either all states or all transitions.

n	12	14	16	18	20	22	24	26	28	30	32
Algo. 1	7.51	6.1	5.0	4.77	4.74	4.24	4.43	4.37	4.34	4.34	4.33
Algo. 2	1.87	1.84	1.83	1.77	1.8	1.78	1.79	1.77	1.77	1.76	1.77
Algo. 3	1	1	1	1	1	1	1	1	1	1	1
Algo. 1	14.1	11.03	10.08	9.27	8.83	9.06	8.64	9.36	9.45	8.59	9.43
Algo. 2	6.9	6.07	5.57	5.08	4.92	4.68	4.66	4.71	4.51	4.52	4.51
Algo. 3	14.1	-	-	9.1	-	-	-	-	8.8	-	-

– Performing100 experiments

– Comparing theoretical and experimental results

Outline

Conclusion and Future Work

On computation times

1. Results obtained using a Python based prototype,
2. Many possible optimizations,
3. Can easily be distributed.

Conclusion

- Algorithms 1 and 2 are tractable.
- Challenging results.
- Algorithm 3 requires more investigations to handle large examples.

Future Work

Perspectives

- Look for Algorithm 3 optimizations,
- Develop an optimized tool,
- Develop efficient algorithms for cleaning the grammar,
- Develop similar approaches for other classes of automata (counter automata, timed automata, ...)