Designing Samplers is Easy: The Boon of Testers

Priyanka Golia[1,2], Mate Soos[2], Sourav Chakraborty[3], and Kuldeep S. Meel[2]

[1]Indian Institute of Technology Kanpur
[2]National University of Singapore
[3]Indian Statistical Institute Kolkata

FMCAD 2021

Testing of Program

- Given a program P and requirement R, test if P satisfies R.

Testing of Program

- Given a program P and requirement R, test if P satisfies R.
- Let requirement R: program P should take two inputs X,Y and outputs maximum of X, Y.

```
int max(X,Y) {
        Return X
}
```

Testing of Program

- Given a program P and requirement R, test if P satisfies R.
- Let requirement R: program P should take two inputs X,Y and outputs maximum of X, Y.

```
int max(X,Y) {
        Return X
}
```

TestCase	Output	Result
$\langle X = 10, Y = 5 \rangle$	10	Pass
$\langle X = 10, Y = 10 \rangle$	10	Pass
$\langle X = 10, Y = 7 \rangle$	10	Pass

All test cases have $X \geq Y$.

- Given a program P and requirement R, test if P satisfies R.
- Let requirement R: program P should take two inputs X,Y and outputs maximum of X, Y.

```
int max(X,Y) {
        Return X
}
```

TestCase	Output	Result
$\langle X = 10, Y = 5\rangle$	10	Pass
$\langle X = 10, Y = 10\rangle$	10	Pass
$\langle X = 5, Y = 10\rangle$	5	Fail

Uniform testcase generator

Uniform Sampling

- Given a Boolean formula F, sample solutions of F uniformly at random.
- Let all samples of F constitute R_F.

$$\forall s \in R_F, Pr[Sampler(F) = s] = \frac{1}{|R_F|}$$

- Let $F = x_1 \vee x_2$

x_1	x_2
0	1
1	0
1	1

R_F: All satisfying assignments of F. $|R_F| = 3$

Generate 2 samples $\longrightarrow$

x_1	x_2
0	1
1	1

Probability of getting each sample = $\frac{1}{3}$

Widespread applications

- Constrained-random simulations. (Naveh, Rimon, Jaeger et al.,2007)
- Constraint-based fuzzing.(Böhme, Heule, Roychoudhury, 2016)
- Configuration testing. (Clarke,1976)
- Bug synthesis. (Roy, Pandey, Dolan-Gavitt,Hu, 2018)
- Functional synthesis. (Golia, Roy, Meel, 2020,2021)

Different Sampling Strategies

Uniform Sampling is computationally hard.

- Knowledge representation based techniques
  (Yuan,Shultz, Pixley,Miller,Aziz 1999)
  (Yuan,Aziz, Pixley,Albin, 2004)
  (Kukula and Shiple, 2000)
  (Sharma, Gupta, Meel, Roy, 2018)
  (Gupta, Sharma, Meel, Roy, 2019)
- Hashing based techniques
  (Chakraborty, Meel, and Vardi 2013, 2014,2015)
  (Soos, Meel, and Gocht 2020)
- Mutation based techniques
  (Dutra, Laeufer, Bachrach, Sen, 2018)
- Markov Chain Monte Carlo based techniques
  (Wei and Selman,2005)
  (Kitchen,2010)
- Constraint solver based techniques
  (Ermon, Gomes, Sabharwal, Selman,2012)
- Belief networks based techniques
  (Dechter, Kask, Bin, Emek,2002)
  (Gogate and Dechter,2006)

Testing of Samplers

- To test whether a sampler is indeed sampling uniformly at random?
- Computation of statistics for generated distributions over small benchmarks.
  - Do not generalize to many classes of benchmarks.
- Barbarik: First scalable sampling tester (Chakraborty and Meel,2019).
  - REJECTs a sampler: the distribution generated by sampler is far from uniform.
  - ACCEPTs a sampler: the distribution generated by sampler is close to uniform under non-adversarial assumption.

Testing of Samplers

- Samplers without guarantees (Uniform-like Samplers):
  - STS (Ermon, Gomes, Sabharwal, Selman,2012)
  - QuickSampler (Dutra, Laeufer, Bachrach, Sen, 2018)
- Sampler with guarantees:
  - UniGen3 (Chakraborty, Meel, and Vardi 2013, 2014,2015)

	QuickSampler	STS	UniGen3
ACCEPTs	0	14	50
REJECTs	50	36	0

Wishlist

How can we use the availability of Barbarik to design a good sampler?

- Sampler should pass the Barbarik test.
- Sampler should be at least as fast as STS and QuickSampler.
- Sampler should perform good on real world applications.

Need of Randomization

- Randomization in the choice of partial assignments.
  - Build partial assignment variable by variable.
  - If partial assignment is incorrect, record, and learn from failure.

Need of Randomization

- Randomization in the choice of partial assignments.
  - Build partial assignment variable by variable.
  - If partial assignment is incorrect, record, and learn from failure.
- Randomized variation of Conflict-Driven Clause Learning (CDCL) framework.
  - Randomized heuristic for what variable to assign next.
  - Randomized heuristic for variable polarities.

CMSGen

- Exploits the flexibility CryptoMiniSat.

CMSGen

- Exploits the flexibility CryptoMiniSat.
- Pick polarities and branch on variables at random.
  - To explore the search space as evenly as possible.
  - To have samples over all the solution space.

CMSGen

- Exploits the flexibility CryptoMiniSat.
- Pick polarities and branch on variables at random.
  - To explore the search space as evenly as possible.
  - To have samples over all the solution space.
- Turn off all pre and inprocessing.
  - Processing techniques: bounded variable elimination, local search, or symmetry breaking, and many more.
  - Can change solution space of instances.

CMSGen

- Exploits the flexibility CryptoMiniSat.
- Pick polarities and branch on variables at random.
  - To explore the search space as evenly as possible.
  - To have samples over all the solution space.
- Turn off all pre and inprocessing.
  - Processing techniques: bounded variable elimination, local search, or symmetry breaking, and many more.
  - Can change solution space of instances.
- Restart at static intervals.
  - Helps to generate samples which are very hard to find.

Power of Test-Driven Development

- Test-Driven Development of CMSGen.
- Parameters of CMSGen are decided with the help of Barbarik
  - Iterative process.
  - Based on feedback from Barbarik, change the parameters.
- Uniform-like-sampler.
- Lack of theoretical analysis.

Testing of Samplers

- Samplers without guarantees (Uniform-like Samplers):
  - STS (Ermon, Gomes, Sabharwal, Selman,2012)
  - QuickSampler (Dutra, Laeufer, Bachrach, Sen, 2018)
- Sampler with guarantees:
  - UniGen3 (Chakraborty, Meel, and Vardi 2013, 2014,2015)

	QuickSampler	STS	UniGen3
ACCEPTs	0	14	50
REJECTs	50	36	0

Testing of Samplers

- Samplers without guarantees (Uniform-like Samplers):
  - STS (Ermon, Gomes, Sabharwal, Selman,2012)
  - QuickSampler (Dutra, Laeufer, Bachrach, Sen, 2018)
  - **CMSGen**
- Sampler with guarantees:
  - UniGen3 (Chakraborty, Meel, and Vardi 2013, 2014,2015)

	QuickSampler	STS	UniGen3	**CMSGen**
ACCEPTs	0	14	50	50
REJECTs	50	36	0	0

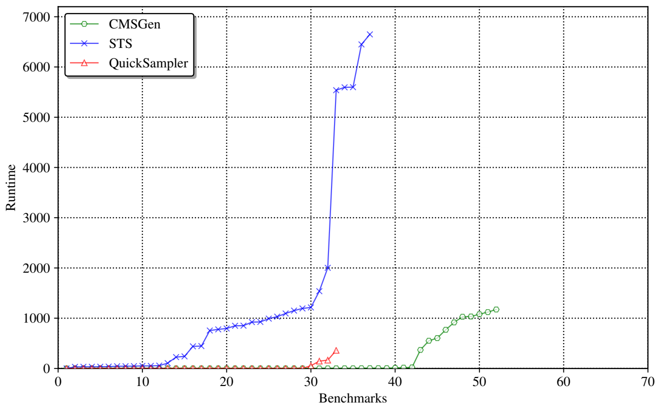
Wishlist

- Sampler should pass the Barbarik test. ✓
- Sampler should be at least as fast as STS and QuickSampler.
- Sampler should perform good on real world applications.

CMSGen vs. Other State-of-the-Art Samplers

- 70 Benchmarks arising from:
  - probabilistic reasoning, (Chakraborty, Fremont, Meel et al.,2015)
  - bounded model checking. (Clarke, Biere, Raimi, Zhu,2001)
  - bug synthesis. (Roy, Pandey, Dolan-Gavitt,Hu, 2018)
- Runtime evaluation to generate 1000 samples.
- Timeout: 7200 seconds.

CMSGen vs. Other State-of-the-Art Samplers (II)



QuickSampler	STS	CMSGen
33	37	52

Wishlist

- Sampler should pass the Barbarik test. ✓
- Sampler should be at least as fast as STS and QuickSampler. ✓
- Sampler should perform good on real world applications.

Case Studies

Usage in application where both scalability and quality are key determining factors.

- Functional Synthesis
- Combinatorial Testing

Functional Synthesis

- Given: A set of inputs (X) and outputs (Y), and underlying specification $\varphi(X, Y)$
- Synthesize: Outputs in terms of inputs such that specification holds.

$$\exists Y \varphi(X, Y) \equiv \varphi(X, F(X))$$

- Objective is to synthesize function $F(X)$
- Wide ranging applications:
  - Logic synthesis (Jiang,Lin, Hung,2009)
  - Program synthesis (Srivastava,Gulwani,Foster,2013)
  - cryptography (Massacci,Marraro,2000)

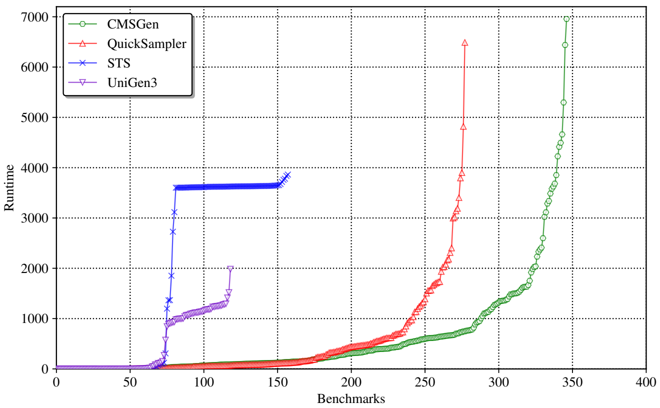
Functional Synthesis (II)

- State-of-the-art approach for Boolean function synthesis, Manthan (Golia, Roy, Meel,2020).
- One of the key component of Manthan: Data-Generation
- Manthan uses constraint sampling to generate the data.

Functional Synthesis (II)

- State-of-the-art approach for Boolean function synthesis, Manthan (Golia, Roy, Meel,2020).
- One of the key component of Manthan: Data-Generation
- Manthan uses constraint sampling to generate the data.
- Experimental Evaluation:
  - Augment Manthan with STS, QuickSampler, UniGen3, and CMSGen.
  - Total benchmarks 609, Timeout: 7200 seconds.

Functional Synthesis: CMSGen vs. Other State-of-the-Art Samplers



UniGen3	STS	QuickSampler	CMSGen
118	157	275	345

Combinatorial Testing

- A powerful paradigm for testing configurable system.
- Challenge: To generate test suites that maximizes t-wise coverage.

$$\text{t-wise coverage:} = \frac{\text{\# of t-sized combinations in test suite}}{\text{all possible valid t-sized combinations}}$$

- To generate the test suites use constraint samplers.
- Uniform sampling to have high t-wise coverage (Plazar, Acher, Perrouin et al., 2019).

Combinatorial Testing

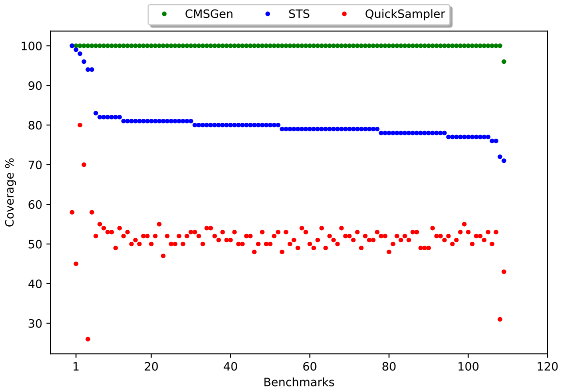
- A powerful paradigm for testing configurable system.
- Challenge: To generate test suites that maximizes t-wise coverage.

$$\text{t-wise coverage:} = \frac{\text{\# of t-sized combinations in test suite}}{\text{all possible valid t-sized combinations}}$$

- To generate the test suites use constraint samplers.
- Uniform sampling to have high t-wise coverage (Plazar, Acher, Perrouin et al., 2019).
- Experimental Evaluations:
  - Generate 1000 samples (test cases).
  - 110 Benchmarks, Timeout: 3600 seconds
  - 2-wise coverage. $t = 2$.

Combinatorial Testing: CMSGen vs. Other State-of-the-Art Samplers

Higher is better

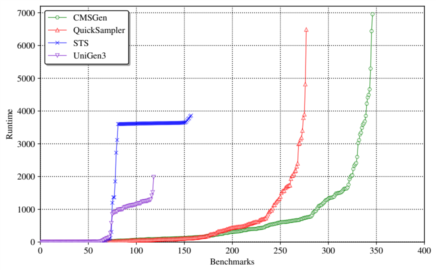


	STS	QuickSampler	CMSGen
Avg. Coverage	80.15%	51.5%	$\sim$ 100%

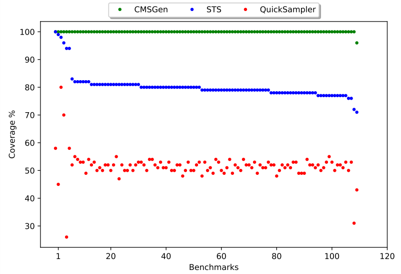
Conclusion

CMSGen: `https://github.com/meelgroup/cmsgen`

	QuickSampler	STS	UniGen3	**CMSGen**
ACCEPTs	0	14	50	50
REJECTs	50	36	0	0



Functional Synthesis



Combinatorial Testing: 2-wise Coverage

Thanks!