Distribution Testing: The New Frontier for Formal Methods

Kuldeep S. Meel

University of Toronto

Joint Adventure with Sourav Chakraborty, Priyanka Golia, Yash Pote, and Mate Soos

Relevant Papers: AAAI-19, FMCAD-21, CP-22, NeurIPS-22

Distribution Testing: The New Frontier for Formal Methods

Kuldeep S. Meel

University of Toronto

Joint Adventure with Sourav Chakraborty, Priyanka Golia, Yash Pote, and Mate Soos

Relevant Papers: AAAI-19, FMCAD-21, CP-22, NeurIPS-22

Summary: A new problem space with opportunities for exciting theory, algorithms, and systems with practical impact.

A Brief (and Biased) History of Computing

Church - Turing Thesis, 1930's: The notion of computability

von Neumann Architecture, 1945: Early hardware implementations

A Brief (and Biased) History of Computing

Church - Turing Thesis, 1930's: The notion of computability

von Neumann Architecture, 1945: Early hardware implementations

Scott-Rabin, 1958: Finite Automaton; the notion of non-determinism in automata

Floyd; Hoare, 1968-69 Assigning meanings to programs; $\{P\}C\{Q\}$

A Brief (and Biased) History of Computing

Church - Turing Thesis, 1930's: The notion of computability

von Neumann Architecture, 1945: Early hardware implementations

Scott-Rabin, 1958: Finite Automaton; the notion of non-determinism in automata

Floyd; Hoare, 1968-69 Assigning meanings to programs; $\{P\}C\{Q\}$

Pneulii, 1977: Introduction of Linear Temporal Logic to CS

Clarke-Emerson; Quielle and Sifakis, 1981 : Birth of Model Checking

The Start of Automated Reasoning Revolution: BDDs, SAT, and Beyond SAT

A Brief (and Biased) History of Computing

Church - Turing Thesis, 1930's: The notion of computability

von Neumann Architecture, 1945: Early hardware implementations

Scott-Rabin, 1958: Finite Automaton; the notion of non-determinism in automata

Floyd; Hoare, 1968-69 Assigning meanings to programs; $\{P\}C\{Q\}$

Pneulii, 1977: Introduction of Linear Temporal Logic to CS

Clarke-Emerson; Quielle and Sifakis, 1981 : Birth of Model Checking

The Start of Automated Reasoning Revolution: BDDs, SAT, and Beyond SAT

Fundamental Aspect: Every execution of the program must satisfy the specification

- A single (or constantly many) execution suffices as witness for falsifiability

Beyond Non-determinism: Power of Randomization

Erdos, 1959: Probabilistic Method in Graph Theory

Solovay and Strassen; Rabin, 1976: Checking primality of a number

Gill, 1977: Computational Complexity of Probabilistic Turing Machines

Beyond Non-determinism: Power of Randomization

Erdos, 1959: Probabilistic Method in Graph Theory

Solovay and Strassen; Rabin, 1976: Checking primality of a number

Gill, 1977: Computational Complexity of Probabilistic Turing Machines

Carter-Wegman, 1977: Strongly Universal Hash Functions

Morris, 1978: Probabilistic Counting

Beyond Non-determinism: Power of Randomization

Erdos, 1959: Probabilistic Method in Graph Theory

Solovay and Strassen; Rabin, 1976: Checking primality of a number

Gill, 1977: Computational Complexity of Probabilistic Turing Machines

Carter-Wegman, 1977: Strongly Universal Hash Functions

Morris, 1978: Probabilistic Counting

And then everything changed in 1980's and world was never the same

Randomization as a Core Ingredient: Distributed Computing, Cryptography, Testing, Streaming, and Machine Learning

With Prevalence comes the opportunity for Formal Methods

How do we test and verify randomness?

- How do we know python's implementation of random is correct?
- How do we know constrained samplers used in testing are generating from desired distributions?

With Prevalence comes the opportunity for Formal Methods

How do we test and verify randomness?

- How do we know python's implementation of random is correct?
- How do we know constrained samplers used in testing are generating from desired distributions?

Challenge: Single (even, constants many) execution do not suffice as witness for falsifiability

- Simple verification problems for probabilistic systems are #P-hard, compared to NP-hardness for (non)-deterministic programs [BGMMPV22]

Is there any hope?

With Prevalence comes the opportunity for Formal Methods

How do we test and verify randomness?

- How do we know python's implementation of random is correct?
- How do we know constrained samplers used in testing are generating from desired distributions?

Challenge: Single (even, constants many) execution do not suffice as witness for falsifiability

- Simple verification problems for probabilistic systems are #P-hard, compared to NP-hardness for (non)-deterministic programs [BGMMPV22]

Is there any hope?

Yes; We can build on the progress in the subfield of distribution testing in theoretical CS community

Distribution Testing: A "subfield, at the junction of property testing and Statistics, is concerned with studying properties of probability distributions." [Canonne, 2020]

Outline

Q1 What do distributions look like in the real world?

Q2 What properties matter to the practitioners?

Q3 How to develop practical scalable testers for distributions?

Q4 Can distribution testing influence the design of systems ?

Q1: Distributions in Real World

Constrained Random Simulation: Test Vector Generation

- Dominant methodology to test hardware systems
- Use a formula φ to encode the verification scenarios
- A Constrained Sampler $\mathcal{A}$ takes φ as input and returns $\sigma \in \mathsf{Sol}(\varphi)$, and ideally ensures

$$\Pr[\sigma \leftarrow \mathcal{A}(\varphi)] = \frac{1}{|\mathsf{Sol}(\varphi)|}$$

Q1: Distributions in Real World

Constrained Random Simulation: Test Vector Generation

- Dominant methodology to test hardware systems
- Use a formula φ to encode the verification scenarios
- A Constrained Sampler $\mathcal{A}$ takes φ as input and returns $\sigma \in \mathsf{Sol}(\varphi)$, and ideally ensures

$$\Pr[\sigma \leftarrow \mathcal{A}(\varphi)] = \frac{1}{|\mathsf{Sol}(\varphi)|}$$

Generative Probabilistic Models: Probabilistic Circuits

- A circuit $\varphi(X, Y)$ where X are input and Y are output
- The resulting distribution over 2^Y when X are assigned values according to prior distribution (say, uniform)

$$\Pr[\sigma \in 2^Y] \propto \#\varphi(X, \sigma)$$

where $\#\varphi(X, \sigma) := \left|\{\rho \in 2^X \mid \{\varphi(\rho, \sigma) = 1\}\}\right|$

Focus of today's talk: Constrained Samplers

Constrained Samplers

- Even finding just a single satisfying assignment is NP-hard
- A well-studied problem by theoreticians and practitioners alike for nearly 40 years
- Only in 2010's, we could have samplers with theoretical guarantees and "reasonable" performance
  - Well, not really reasonable from practical perspective
- Design of *practical* samplers based on MCMC, random walk, local search etc.

Constrained Samplers

- Even finding just a single satisfying assignment is NP-hard
- A well-studied problem by theoreticians and practitioners alike for nearly 40 years
- Only in 2010's, we could have samplers with theoretical guarantees and "reasonable" performance
  - Well, not really reasonable from practical perspective
- Design of *practical* samplers based on MCMC, random walk, local search etc.
- Three Samplers that we will consider in our talk
  - UniGen3: Theoretical Guarantees of almost-uniformity [CMV13; CMV14; SGM20]
  - SearchTreeSampler: Very weak guarantees [EGSS12]
  - QuickSampler: No Guarantees [DLBS18]
- The study (in 2018) that proposed Quicksampler could only perform unsound statistical tests, and therefore, could not distinguish the three samplers

Goal: Develop sound procedures to distinguish samplers (if possible).

Outline

Q1 What do distributions look like in the real world?

Q2 What properties matter to the practitioners?

Q3 How to develop practical scalable testers for distributions?

Q4 Can distribution testing influence the design of systems ?

Q2: Properties that Matter

(Approximate) Equivalence Checking

- (Fast) Sampler $\mathcal{A}$ and a reference (but, often slow) sampler $\mathcal{U}$
- Reference sampler $\mathcal{U}$ is certified to produce samples according to desired distribution but is slow.
- Is the distribution generated by $\mathcal{A}$, denoted by $\mathcal{A}_\varphi$, close to that of $\mathcal{U}_\varphi$?

Q2: Properties that Matter

(Approximate) Equivalence Checking

- (Fast) Sampler $\mathcal{A}$ and a reference (but, often slow) sampler $\mathcal{U}$
- Reference sampler $\mathcal{U}$ is certified to produce samples according to desired distribution but is slow.
- Is the distribution generated by $\mathcal{A}$, denoted by $\mathcal{A}_\varphi$, close to that of $\mathcal{U}_\varphi$?

Support Size Estimation

- Given a Distribution $\mathcal{P}$, compute the size of $|\{\sigma \mid \mathcal{P}(\sigma) > 0\}$

Q2: Properties that Matter

(Approximate) Equivalence Checking

- (Fast) Sampler $\mathcal{A}$ and a reference (but, often slow) sampler $\mathcal{U}$
- Reference sampler $\mathcal{U}$ is certified to produce samples according to desired distribution but is slow.
- Is the distribution generated by $\mathcal{A}$, denoted by $\mathcal{A}_\varphi$, close to that of $\mathcal{U}_\varphi$?

Support Size Estimation

- Given a Distribution $\mathcal{P}$, compute the size of $|\{\sigma \mid \mathcal{P}(\sigma) > 0\}$

Quantified Information Flow

- Given a circuit $\varphi(X, Y)$ (where X are input and Y are output), compute the entropy of the output distribution:

This Talk's Focus: Equivalence

Consider two distribution $\mathcal{P}$ and $\mathcal{Q}$ over $\{0,1\}^n$.

Two Notions of Distance

- d_∞ distance: $\max_{\sigma \in \{0,1\}^n} |\mathcal{P}(\sigma) - \mathcal{Q}(\sigma)|$
  - The most commonly seen behavior where a developer wants to approximate $\mathcal{P}$ with another distribution $\mathcal{Q}$
  - Almost-uniform sampling in the context of constrained random simulation

This Talk's Focus: Equivalence

Consider two distribution $\mathcal{P}$ and $\mathcal{Q}$ over $\{0,1\}^n$.

Two Notions of Distance

- d_∞ distance: $\max_{\sigma \in \{0,1\}^n} |\mathcal{P}(\sigma) - \mathcal{Q}(\sigma)|$
  - The most commonly seen behavior where a developer wants to approximate $\mathcal{P}$ with another distribution $\mathcal{Q}$
  - Almost-uniform sampling in the context of constrained random simulation
- Total Variation Distance (d_{TV}) or L_1 distance: $\frac{1}{2} \sum_{\sigma \in \{0,1\}^n} |\mathcal{P}(\sigma) - \mathcal{Q}(\sigma)|$
  - Consider any arbitrary program $\mathcal{A}$ that uses samples from a distribution: there is a probability distribution over output of $\mathcal{A}$.
  - Consider a Bad event over the output of $\mathcal{A}$: such as not catching a bug.
  - Let's say $\mathcal{A}$ samples from $\mathcal{P}$.
  - Folklore: If we were to replace $\mathcal{P}$ with $\mathcal{Q}$ then the probability of Bad event would increase/decrease at most by $d_{TV}(P, Q)$.

This Talk's Focus: Equivalence

Consider two distribution $\mathcal{P}$ and $\mathcal{Q}$ over $\{0,1\}^n$.

Two Notions of Distance

- d_∞ distance: $\max_{\sigma \in \{0,1\}^n} |\mathcal{P}(\sigma) - \mathcal{Q}(\sigma)|$
  - The most commonly seen behavior where a developer wants to approximate $\mathcal{P}$ with another distribution $\mathcal{Q}$
  - Almost-uniform sampling in the context of constrained random simulation
- Total Variation Distance (d_{TV}) or L_1 distance: $\frac{1}{2} \sum_{\sigma \in \{0,1\}^n} |\mathcal{P}(\sigma) - \mathcal{Q}(\sigma)|$
  - Consider any arbitrary program $\mathcal{A}$ that uses samples from a distribution: there is a probability distribution over output of $\mathcal{A}$.
  - Consider a Bad event over the output of $\mathcal{A}$: such as not catching a bug.
  - Let's say $\mathcal{A}$ samples from $\mathcal{P}$.
  - Folklore: If we were to replace $\mathcal{P}$ with $\mathcal{Q}$ then the probability of Bad event would increase/decrease at most by $d_{TV}(P, Q)$.

Therefore, measure closeness with respect to d_∞ and farness with respect to d_{TV}

- Checker should return Accept if two distributions are close in d_∞-distance and return Reject if two distributions are far in d_{TV}.

Problem Setting

- A Boolean formula φ
- Reference Sampler $\mathcal{U}$
  - With rigorous theoretical guarantees but often slower
- Sampler Under Test: A sampler $\mathcal{A}$ that claims to be close to uniform sampler for formulas in benchmark set
  - Superior runtime performance but often no theoretical analysis
- Closeness and farness parameters: ε and η

Task: Determine whether distributions $\mathcal{A}_\varphi$ and $\mathcal{U}_\varphi$ are ε-close or η-far

Outline

Q1 What do distributions look like in the real world?

Q2 What properties matter to the practitioners?

Q3 How to develop practical scalable testers for distributions?

Q4 Can distribution testing influence the design of systems ?

Limitations of Black-Box Testing

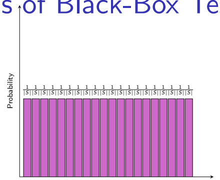


Figure: $\mathcal{U}_\varphi$: Uniform Distribution

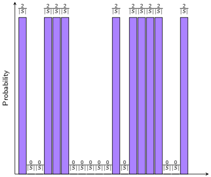


Figure: $\mathcal{A}_\varphi$: 1/2-far from uniform

SAMP: Allows you to draw samples from a distribution

Limitations of Black-Box Testing

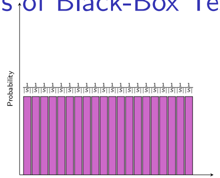


Figure: $\mathcal{U}_\varphi$: Uniform Distribution

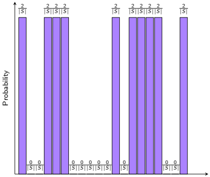


Figure: $\mathcal{A}_\varphi$: 1/2-far from uniform

SAMP: Allows you to draw samples from a distribution

- If $< \sqrt{|\mathrm{Sol}(\varphi)|}/100$ samples are drawn then with high probability you see only distinct samples from either distribution.

Theorem The above bound is optimal. [BFRSW 98; Pan 08]

Limitations of Black-Box Testing

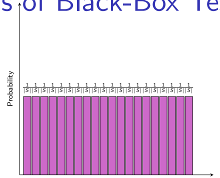


Figure: $\mathcal{U}_\varphi$: Uniform Distribution

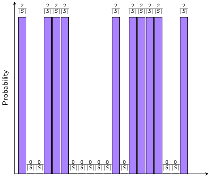


Figure: $\mathcal{A}_\varphi$: 1/2-far from uniform

SAMP: Allows you to draw samples from a distribution

- If $< \sqrt{|\mathsf{Sol}(\varphi)|}/100$ samples are drawn then with high probability you see only distinct samples from either distribution.

Theorem The above bound is optimal. [BFRSW 98; Pan 08]

Greybox Testing: Inspired by Distribution Testing Literature

COND $(\mathcal{P}, T)$

$$\Pr[\sigma \leftarrow \mathsf{COND}(\mathcal{P}, T)] = \begin{cases} \frac{\mathcal{P}(\sigma)}{\sum_{\rho \in T} \mathcal{P}(\rho)} & \sigma \in T \\ 0 & \text{otherwise} \end{cases}$$

When $T = \{0,1\}^n$, then $\mathsf{COND}(\mathcal{P}, T) = \mathsf{SAMP}$

The Power of COND model

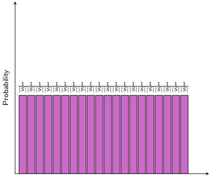


Figure: $\mathcal{U}_\varphi$: Uniform Distribution

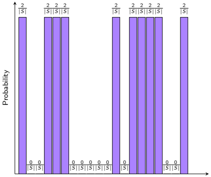


Figure: $\mathcal{A}_\varphi$: 1/2-far from uniform

The Power of COND model

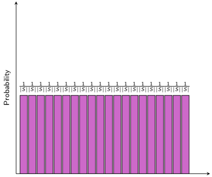


Figure: $\mathcal{U}_\varphi$: Uniform Distribution

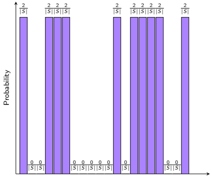


Figure: $\mathcal{A}_\varphi$: 1/2-far from uniform

An algorithm for testing uniformity using conditional sampling:

- Sample σ_1 from $\mathcal{U}_\varphi$ and σ_2 from $\mathcal{A}_\varphi$. Let $T = \{\sigma_1, \sigma_2\}$.
- In the case of the "far" distribution, with constant probability, $\mathcal{A}_\varphi(\sigma_1) \ll \mathcal{A}_\varphi(\sigma_2)$
- We will be able to distinguish the far distribution from the uniform distribution using constant number of samples from COND($\mathcal{A}_\varphi$, T).
- The constant depend on the farness parameter.

From Theory to Practice: Realizing COND Model

Challenge: How do we ask sampler for Conditional samples over $T = \{\sigma_1, \sigma_2\}$.

- Construct $\hat{\varphi} = \varphi \wedge (X = \sigma_1 \vee X = \sigma_2)$

From Theory to Practice: Realizing COND Model

Challenge: How do we ask sampler for Conditional samples over $T = \{\sigma_1, \sigma_2\}$.

- Construct $\hat{\varphi} = \varphi \wedge (X = \sigma_1 \vee X = \sigma_2)$

Almost all the constrained samplers just enumerate all the solutions when the number of solutions is small

From Theory to Practice: Realizing COND Model

Challenge: How do we ask sampler for Conditional samples over $T = \{\sigma_1, \sigma_2\}$.

- Construct $\hat{\varphi} = \varphi \wedge (X = \sigma_1 \vee X = \sigma_2)$

Almost all the constrained samplers just enumerate all the solutions when the number of solutions is small

- Need way to construct formulas whose solution space is large but every solution can be mapped to either σ_1 or σ_2.

Kernel

Input: A Boolean formula φ, two assignments σ_1 and σ_2, and desired number of solutions τ
Output: Formula $\hat{\varphi}$

- $\tau = |\mathsf{Sol}(\hat{\varphi})|$
- $z \in \mathsf{Sol}(\hat{\varphi}) \implies z_{\downarrow X} \in \{\sigma_1, \sigma_2\}$
- $|\{z \in \mathsf{Sol}(\hat{\varphi}) \mid z_{\downarrow X} = \sigma_1\}| = |\{z \in \mathsf{Sol}(\hat{\varphi}) \mid z_{\downarrow X} \cap \sigma_2\}|$
- φ and $\hat{\varphi}$ has similar structure

Kernel

Input: A Boolean formula φ, two assignments σ_1 and σ_2, and desired number of solutions τ
Output: Formula $\hat{\varphi}$

- $\tau = |\mathsf{Sol}(\hat{\varphi})|$
- $z \in \mathsf{Sol}(\hat{\varphi}) \implies z_{\downarrow X} \in \{\sigma_1, \sigma_2\}$
- $|\{z \in \mathsf{Sol}(\hat{\varphi}) \mid z_{\downarrow X} = \sigma_1\}| = |\{z \in \mathsf{Sol}(\hat{\varphi}) \mid z_{\downarrow X} \cap \sigma_2\}|$
- φ and $\hat{\varphi}$ has similar structure

Non-adversarial Sampler Assumption: The distribution of the projection of samples obtained from $\mathcal{A}_{\hat{\varphi}}$ to variables of φ is same as $\mathsf{COND}(\mathcal{A}_{\varphi}, \{\sigma_1, \sigma_2\})$.

Implications:

- If $\mathcal{A}$ is a uniform sampler for every Boolean formula, it satisfies non-adversarial sampler assumption
- If $\mathcal{A}$ is not a uniform sampler, it may not necessarily satisfy non-adversarial sampler assumption

Kernel

Input: A Boolean formula φ, two assignments σ_1 and σ_2, and desired number of solutions τ
Output: Formula $\hat{\varphi}$

- $\tau = |\mathsf{Sol}(\hat{\varphi})|$
- $z \in \mathsf{Sol}(\hat{\varphi}) \implies z_{\downarrow X} \in \{\sigma_1, \sigma_2\}$
- $|\{z \in \mathsf{Sol}(\hat{\varphi}) \mid z_{\downarrow X} = \sigma_1\}| = |\{z \in \mathsf{Sol}(\hat{\varphi}) \mid z_{\downarrow X} \cap \sigma_2\}|$
- φ and $\hat{\varphi}$ has similar structure

Non-adversarial Sampler Assumption: The distribution of the projection of samples obtained from $\mathcal{A}_{\hat{\varphi}}$ to variables of φ is same as $\mathsf{COND}(\mathcal{A}_{\varphi}, \{\sigma_1, \sigma_2\})$.

Implications:

- If $\mathcal{A}$ is a uniform sampler for every Boolean formula, it satisfies non-adversarial sampler assumption
- If $\mathcal{A}$ is not a uniform sampler, it may not necessarily satisfy non-adversarial sampler assumption

Non-adversarial assumption allows us to use the theory of COND query model

Barbarik

Input: A sampler under test $\mathcal{A}$, a reference uniform sampler $\mathcal{U}$, a tolerance parameter $\varepsilon > 0$, an intolerance parmaeter $\eta > \varepsilon$, a guarantee parameter δ and a CNF formula φ

Output: ACCEPT or REJECT with the following guarantees:

- if the generator $\mathcal{A}$ is ε-close (in d_∞), i.e., $d_\infty(\mathcal{A},\mathcal{U}) \leq \varepsilon$ then Barbarik ACCEPTS with probability at least $(1 - \delta)$.
- If the generator $\mathcal{A}$ is η-far in (d_{TV}), i.e., $d_{TV}(\mathcal{A},\mathcal{U}) > \eta$ and if non-adversarial sampler assumption holds then Barbarik REJECTS with probability at least $1 - \delta$.

Observe: Complexity independent of $|\mathrm{Sol}(\mathit{varphi})|$ in contrast to black box's approach's dependence on $\sqrt{|\mathrm{Sol}(\mathit{varphi})|}$

Barbarik

Input: A sampler under test $\mathcal{A}$, a reference uniform sampler $\mathcal{U}$, a tolerance parameter $\varepsilon > 0$, an intolerance parmaeter $\eta > \varepsilon$, a guarantee parameter δ and a CNF formula φ

Output: ACCEPT or REJECT with the following guarantees:

- if the generator $\mathcal{A}$ is ε-close (in d_∞), i.e., $d_\infty(\mathcal{A},\mathcal{U}) \leq \varepsilon$ then Barbarik ACCEPTS with probability at least $(1-\delta)$.
- If the generator $\mathcal{A}$ is η-far in (d_{TV}), i.e., $d_{TV}(\mathcal{A},\mathcal{U}) > \eta$ and if non-adversarial sampler assumption holds then Barbarik REJECTS with probability at least $1-\delta$.

Observe: Complexity independent of $|\mathsf{Sol}(varphi)|$ in contrast to black box's approach's dependence on $\sqrt{|\mathsf{Sol}(varphi)|}$

Experimental Evaluation over three state of the art (almost-)uniform samplers

- UniGen3: Theoretical Guarantees of almost-uniformity
- SearchTreeSampler: Very weak guarantees
- QuickSampler: No Guarantees

The study (in 2018) that proposed Quicksampler could only perform unsound statistical tests, and therefore, could not distinguish the three samplers

Results-I

Instances	#Solutions	UniGen3		SearchTreeSampler	
		Output	#Samples	Output	#Samples
71	1.14×2^{59}	A	1729750	R	250
blasted_case49	1.00×2^{61}	A	1729750	R	250
blasted_case50	1.00×2^{62}	A	1729750	R	250
scenarios_aig_insertion1	1.06×2^{65}	A	1729750	R	250
scenarios_aig_insertion2	1.06×2^{65}	A	1729750	R	250
36	1.00×2^{72}	A	1729750	R	250
30	1.73×2^{72}	A	1729750	R	250
110	1.09×2^{76}	A	1729750	R	250
scenarios_tree_insert_insert	1.32×2^{76}	A	1729750	R	250
107	1.52×2^{76}	A	1729750	R	250
blasted_case211	1.00×2^{80}	A	1729750	R	250
blasted_case210	1.00×2^{80}	A	1729750	R	250
blasted_case212	1.00×2^{88}	A	1729750	R	250
blasted_case209	1.00×2^{88}	A	1729750	R	250
54	1.15×2^{90}	A	1729750	R	250

Results-II

Instances	#Solutions	UniGen3		QuickSampler	
		Output	#Samples	Output	#Samples
71	1.14×2^{59}	A	1729750	R	250
blasted_case49	1.00×2^{61}	A	1729750	R	250
blasted_case50	1.00×2^{62}	A	1729750	R	250
scenarios_aig_insertion1	1.06×2^{65}	A	1729750	R	250
scenarios_aig_insertion2	1.06×2^{65}	A	1729750	R	250
36	1.00×2^{72}	A	1729750	R	250
30	1.73×2^{72}	A	1729750	R	250
110	1.09×2^{76}	A	1729750	R	250
scenarios_tree_insert_insert	1.32×2^{76}	A	1729750	R	250
107	1.52×2^{76}	A	1729750	R	250
blasted_case211	1.00×2^{80}	A	1729750	R	250
blasted_case210	1.00×2^{80}	A	1729750	R	250
blasted_case212	1.00×2^{88}	A	1729750	R	250
blasted_case209	1.00×2^{88}	A	1729750	R	250
54	1.15×2^{90}	A	1729750	R	250

Outline

Q1 What do distributions look like in the real world?

Q2 What properties matter to the practitioners?

Q3 What are the resource constraints?

Q4 Can distribution testing influence the design of systems?

Outline

Q1 What do distributions look like in the real world?

Q2 What properties matter to the practitioners?

Q3 What are the resource constraints?

Q4 Can distribution testing influence the design of systems?

Wishlist

- Sampler should be at least as fast as STS and QuickSampler.
- Sampler should by accepted by Barbarik.
- Sampler should have impact on downstream (real world) applications.

CMSGen

- Exploits the flexibility CryptoMiniSat.

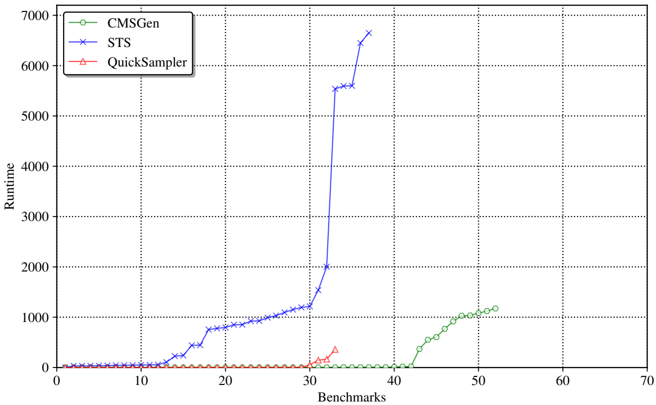
CMSGen

- 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.
- 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 Distribution Testing-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
  - We have very little idea about why SAT solvers work?
  - Much less about what happens when you tweak them to make them samplers

Runtime Performance



QuickSampler	STS	CMSGen
33	37	52

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 [EGSS12]
  - QuickSampler [DLBS18]
  - **CMSGen**
- Sampler with guarantees:
  - UniGen3 [CMV13, CMV14, SGM20]

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

Outline

Q1 What do distributions look like in the real world?

Q2 What properties matter to the practitioners?

Q3 What are the resource constraints?

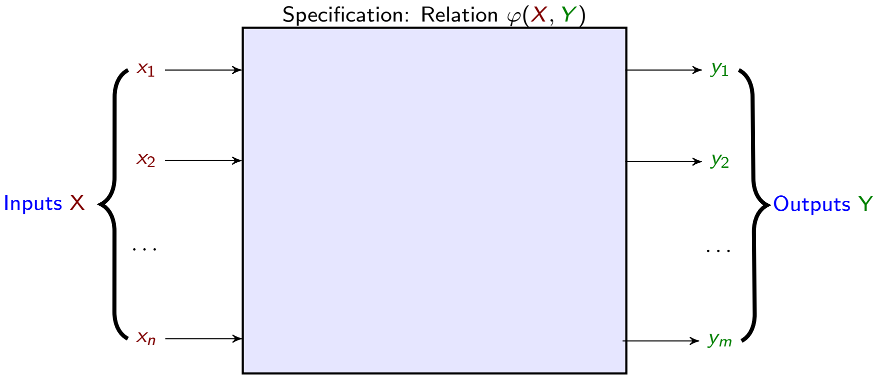
Q4 Can distribution testing influence the design of systems?

Wishlist

- Sampler should be at least as fast as STS and QuickSampler. ✓
- Sampler should by accepted by Barbarik. ✓
- Sampler should have impact on downstream (real world) applications.

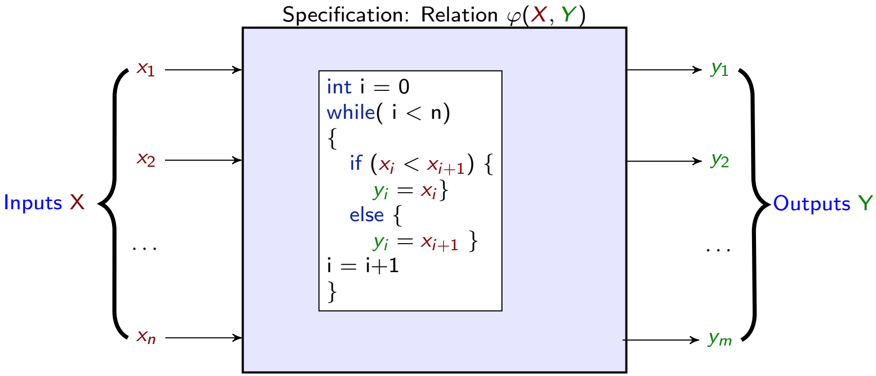
Application I: Functional Synthesis

Holy Grail of Programming: *The user states the problem, the computer solves it* (Freuder, 1996)



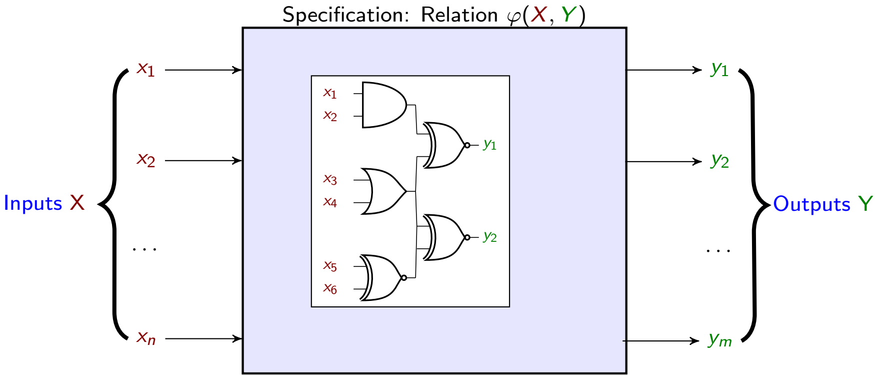
Application I: Functional Synthesis

Holy Grail of Programming: *The user states the problem, the computer solves it* (Freuder, 1996)



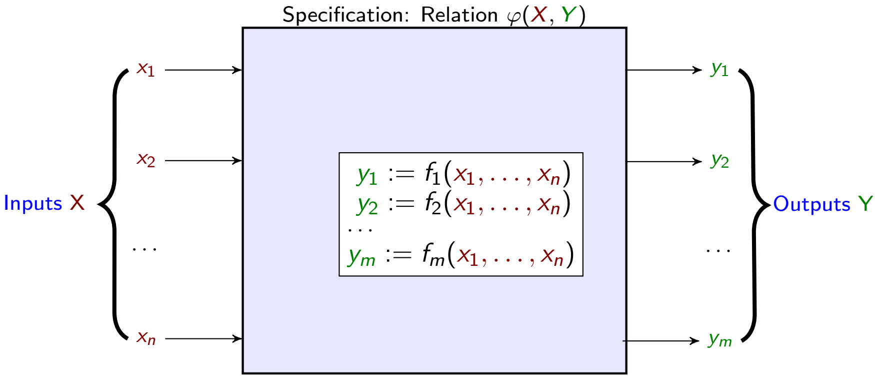
Application I: Functional Synthesis

Holy Grail of Programming: *The user states the problem, the computer solves it* (Freuder, 1996)



Application I: Functional Synthesis

Holy Grail of Programming: *The user states the problem, the computer solves it* (Freuder, 1996)



Application I: Functional Synthesis

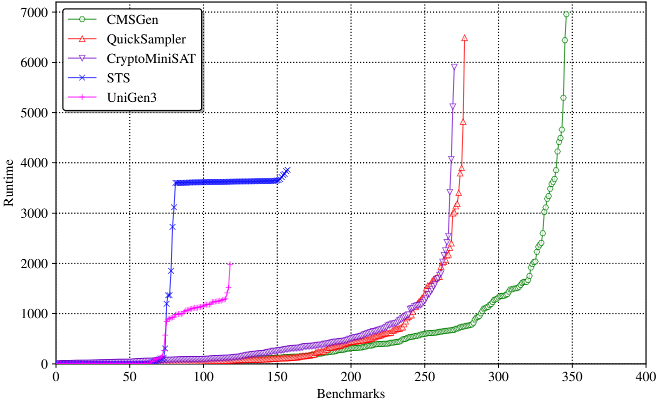
State of the art approach: Manthan

Sampling + Machine Learning + Counter-example guided repair

Application I: Functional Synthesis

State of the art approach: Manthan

Sampling + Machine Learning + Counter-example guided repair



Application II: 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.

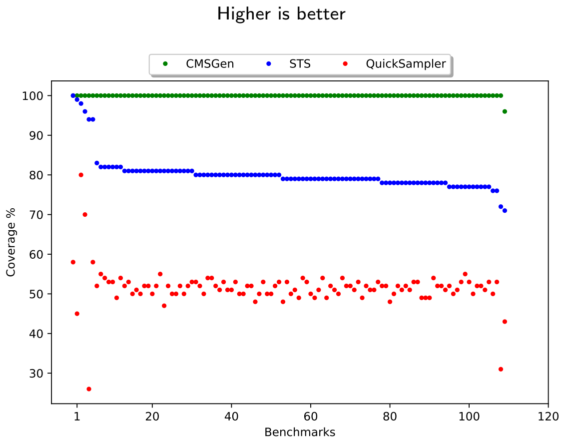
Application II: Combinatorial Testing

- 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.
- Experimental Evaluations:
  - Generate 1000 samples (test cases).
  - 110 Benchmarks, Timeout: 3600 seconds
  - 2-wise coverage $t = 2$.

Combinatorial Testing: The Power of CMSGen



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

Outline

Q1 What do distributions look like in the real world?

Q2 What properties matter to the practitioners?

Q3 What are the resource constraints?

Q4 Can distribution testing influence the design of systems?

Wishlist

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

Conclusion

Q1 What do distributions look like in the real world?

Ans Probability distributions are first-class objects in modern computing

Q2 What properties matter to the practitioners?

Ans Equivalence, Support Size Estimation, Entropy

Q3 How to develop practical scalable testers for distributions?

Ans Greybox access, which can be modeled via Conditional Sampling

Q4 Can distribution testing influence the design of systems ?

Ans Yes. It can allow us to design state of the art samplers via a different approach. And such samplers dramatically improve downstream applications.

Where do we go from here?

We have just started!

- Scalable testers for distributions beyond uniform
- Scalable samplers for SMT/CSP via Test-Driven Development
- Developing the notion of counterexample for testing distributions
- How do we certify the correctness of distribution testers?

CMSGen (MIT License): `https://github.com/meelgroup/cmsgen`

Barbarik (MIT License): `https://github.com/meelgroup/barbarik`

These slides are available at `https://www.cs.toronto.edu/~meel/talks.html`