Sampling from combinatorial spaces: Achieving the fine balancing act between independence and scalability

Kuldeep Meel
Rice University

Joint work with Supratik Chakraborty, Daniel J. Fremont, Sanjit A. Seshia, Moshe Y. Vardi
How do we guarantee that systems work correctly?

Functional Verification

- Formal verification
  - Challenges: formal requirements, scalability
  - ~10-15% of verification effort

- Dynamic verification: dominant approach
Dynamic Verification

- Design is simulated with test vectors
- Test vectors represent different verification scenarios
- Results from simulation compared to intended results
- **Challenge**: Exceedingly large test space!
Motivating Example

\[ c = f(a, b) \]

How do we test the circuit works?

- Try for all values of \( a \) and \( b \)
  - \( 2^{128} \) possibilities
  - Sun will go nova before done!
  - Not scalable
Constrained-Random Simulation

Sources for Constraints

• Designers:
  1. \(a +_{64} 11 \times_{32} b = 12\)
  2. \(a <_{64} (b >> 4)\)

• Past Experience:
  1. \(40 <_{64} 34 + a <_{64} 5050\)
  2. \(120 <_{64} b <_{64} 230\)

• Users:
  1. \(232 \times_{32} a + b != 1100\)
  2. \(1020 <_{64} (b /_{64} 2) +_{64} a <_{64} 2200\)

- Test vectors: solutions of constraints
- Proposed by Lichtenstein, Malka, Aharon (IA\(^5\)AI 94)
Problem: How can we *uniformly* sample the values of a and b satisfying the above constraints?
Problem Formulation

Set of Constraints

Sample satisfying assignments uniformly at random

SAT Formula

c = f(a, b)

a

64 bit

b

64 bit

c

64 bit

Scalable Uniform Generation of SAT Witnesses
Prior Work

Guarantees

BGP  BDD

Performance

MCMC  SAT-Based
## EDA Industry’s Desired Performance

<table>
<thead>
<tr>
<th>Generator</th>
<th>Relative Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>XORSample’ (weak guarantees)</td>
<td>~50000</td>
</tr>
<tr>
<td>Desired Uniform Generator*</td>
<td>10</td>
</tr>
<tr>
<td>Simple SAT solver</td>
<td>1</td>
</tr>
</tbody>
</table>
Our Contribution

Performance Guarantees

BGP  BDD  

UniWit  UniGen  UniGen2

CAV 13  DAC 14  TACAS 15

MCMC  SAT-Based
Outline

- Losing Independence of hashing functions
- Losing Independence among samples
- Parallelization of Constrained Random Simulation
- Conclusion
Main Idea
Partitioning into cells

Cells should be roughly equal in size and small enough to enumerate completely.
Partitioning into cells

- Too large => Hard to enumerate
- Too small => Variance can be very high
- hiThresh: upper bound on size of cell
- loThresh: lower bound on size of cell
  - E.g., loThresh = 11, hiThresh = 60
Partitioning into cells
Partitioning into cells

Pick a random cell

Pick a solution randomly from this cell
Partitioning into cells

How can we partition into roughly equal small cells without knowing the distribution of solutions?

Universal Hashing
(Carter-Wegman 1979)
Universal Hashing

- Hash functions: mapping \( \{0,1\}^n \) to \( \{0,1\}^m \)
  - \( (2^n \text{ elements to } 2^m \text{ cells}) \)
- Random inputs => All cells are *roughly* equal (in expectation)

- Universal family of hash functions:
  - Choose hash function randomly from family
  - For *arbitrary* distribution on inputs => All cells are *roughly* equal (in expectation)
r-Universal Hashing

- Each solution is hashed uniformly
- Every $r$-subset of solutions is hashed independently
- For $r=2$,

\[ \forall \text{ distinct } y_1, y_2 \text{ and } \forall \alpha_1, \alpha_2 \]
\[ Pr[h(y_1) = \alpha_1 \wedge h(y_2) = \alpha_2] = Pr[h(y_1) = \alpha_1]Pr[h(y_2) = \alpha_2] \]

- $r$-wise universal hash function $\Rightarrow$ polynomial of degree $r-1$
Why Independence matters?

We pick a random cell and define following random variables

$$I_k = 1 \text{ if } y_k \text{ is in the cell}$$

Let $I_1, I_2, I_3, \ldots I_n$ be r-wise independent variables in $[0, 1]$, then for $I = \sum I_k$

$$Pr[|I - \mu| < \delta \mu] \geq c^{-r}$$

Number of solutions in a randomly picked cell

Deviation
Tradeoff

Higher universality => Stronger Guarantees

Higher universality => Polynomials of higher degree
Prior Work

- Choose \( n \)-wise universal hash functions and all cells are guaranteed to be small with high probability
- Theoretical guarantee of uniformity
What if we lower the hashing to 3-universal

All cells are not guaranteed to be small anymore with high probability
What if we lower the hashing to 3-universal

But a randomly picked cell is guaranteed to be small with high probability
Guarantees of almost-uniformity
Strong Theoretical Guarantees

- Uniformity (BGP with n-universal)
  \[ \Pr[y \text{ is output}] = \frac{1}{|R_F|} \]

- Almost-Uniformity (UniGen with 3-universal)
  \[ \forall y \in R_F, \frac{1}{(1 + \varepsilon)|R_F|} \leq \Pr[y \text{ is output}] \leq (1 + \varepsilon)\frac{1}{|R_F|} \]

- Polynomial number of SAT calls
Enumerating cell solutions

- A cell can be represented as the conjunction of:
  - Input formula F
  - \( m \) random XOR constraints
- \( 2^m \) is the number of cells desired

- Use CryptoMiniSAT for CNF + XOR formulas
2-3 Orders of Magnitude Faster

Timeout: 18000 seconds

<table>
<thead>
<tr>
<th>Benchmarks</th>
<th>UniGen</th>
<th>XORSample'</th>
</tr>
</thead>
<tbody>
<tr>
<td>case47</td>
<td></td>
<td></td>
</tr>
<tr>
<td>case3_b14_3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>case105</td>
<td></td>
<td></td>
</tr>
<tr>
<td>case8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>case203</td>
<td></td>
<td></td>
</tr>
<tr>
<td>case145</td>
<td></td>
<td></td>
</tr>
<tr>
<td>case61</td>
<td></td>
<td></td>
</tr>
<tr>
<td>case9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>case15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>case140</td>
<td></td>
<td></td>
</tr>
<tr>
<td>case2_b14_1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>case3_b14_1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>squaring14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>squaring7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>case2_ptb_1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>case1_ptb_1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>case2_b14_2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>case3_b14_2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Time(s)
## Runtime Performance

Experiments over 200+ benchmarks

<table>
<thead>
<tr>
<th>Generator</th>
<th>Relative Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>XORSample’ (weak guarantees)</td>
<td>~50000</td>
</tr>
<tr>
<td>UniGen</td>
<td>470</td>
</tr>
<tr>
<td>Desired Uniform Generator*</td>
<td>10</td>
</tr>
<tr>
<td>Simple SAT solver</td>
<td>1</td>
</tr>
</tbody>
</table>

*: Based on EDA Industry
Outline

- Losing Independence of hashing functions

- Losing Independence among samples

- Parallelization of Constrained Random Simulation

- Conclusion
How many solutions are generated per sample?

> LoThresh
Pick a random cell and check if it's small.

Pick a solution randomly from this cell.

The number of solutions in a small cell is between loThresh and hiThresh.
UniGen

Pick a random cell and check if its small

Pick a lowThresh solutions randomly from this cell

# of solutions in a small cell is between lowThresh and highThresh
3-Universal and Independence of Samples

3-Universal hash functions:

- Choose hash function randomly
- For arbitrary distribution on solutions $\Rightarrow$ All cells are *roughly* equal in expectation

- But:
  - While each input is hashed *uniformly*
  - And each 3-solutions set is hashed *independently*
  - A 4-solutions set *might not* be hashed *independently*
3-Universal and Independence of samples

- Choosing up to 3 samples => Full Independence between samples
- Independence provides coverage guarantees.
Choosing up to 3 samples => Full Independence between samples

Choosing loThresh (> 3) samples => Loss of full independence among samples
- “Almost-Independence”
- Still provides theoretical guarantees of coverage
Strong Guarantees

- \( L = \# \text{ of samples} < |R_F| \)

\[ \forall y \in R_F, \quad \frac{L}{(1 + \varepsilon)|R_F|} \leq \Pr[y \text{ is output}] \leq 1.02(1 + \varepsilon) \frac{L}{|R_F|} \]

- \textbf{Polynomial} Constant number of SAT calls per sample
Bug-finding effectiveness

bug frequency $f = \frac{B}{R_F}$

<table>
<thead>
<tr>
<th>relative number of SAT calls</th>
<th>UniGen</th>
<th>UniGen2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\frac{3 \cdot hi\text{Thresh}(1+\nu)(1+\varepsilon)}{0.52}$</td>
<td>$\frac{3 \cdot hi\text{Thresh}}{0.62 \cdot lo\text{Thresh}} \cdot \frac{(1+\hat{\nu})(1+\varepsilon)}{1-\hat{\nu}}$</td>
</tr>
</tbody>
</table>

Simply put, #of SAT calls for UniGen2 $<<$ # of SAT calls for UniGen.
Bug-finding effectiveness

bug frequency $f = 1/10^4$
find bug with probability $\geq 1/2$

<table>
<thead>
<tr>
<th></th>
<th>UniGen</th>
<th>UniGen2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expected number of SAT calls</td>
<td>$4.35 \times 10^7$</td>
<td>$3.38 \times 10^6$</td>
</tr>
</tbody>
</table>

An order of magnitude difference!
~20 times faster than UniGen

Time per sample (s)

Benchmarks

UniGen2

UniGen
# Runtime Performance

Experiments over 200+ benchmarks

<table>
<thead>
<tr>
<th>Generator</th>
<th>Relative Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>XORSample’ (weak guarantees)</td>
<td>~50000</td>
</tr>
<tr>
<td>UniGen</td>
<td>470</td>
</tr>
<tr>
<td>UniGen2</td>
<td>21</td>
</tr>
<tr>
<td>Desired Uniform Generator*</td>
<td>10</td>
</tr>
<tr>
<td>Simple SAT solver</td>
<td>1</td>
</tr>
</tbody>
</table>

*: Based on EDA Industry
Outline

- Losing Independence of hashing functions
- Losing Independence among samples
- **Parallelization of Constrained Random Simulation**
- Conclusion
Current Paradigm of Simulation-based Verification

- Can not be parallelized since test generators maintain “global state”
- Loses theoretical guarantees (if any) of uniformity
New Paradigm of Simulation-based Verification

- Preprocessing needs to be done only once
- No communication required between different copies of the test generator
- Scales linearly with number of cores in practice
## Desired Performance with 2 cores

<table>
<thead>
<tr>
<th>Generator</th>
<th>Relative Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>UniGen</td>
<td>470</td>
</tr>
<tr>
<td>UniGen2</td>
<td>21</td>
</tr>
<tr>
<td>Parallel UniGen2 (2 cores)</td>
<td>~10</td>
</tr>
<tr>
<td>Desired Uniform Generator*</td>
<td>10</td>
</tr>
<tr>
<td>Simple SAT solver</td>
<td>1</td>
</tr>
</tbody>
</table>

* Based on EDA Industry
Uniformity Comparison

- Benchmark with 16,384 solutions
- Ideal Generator: Enumerate all solutions and pick one randomly
- Generated 4M samples for Ideal, UniGen2 & parallel (on 12 cores) UniGen2
- Group solutions according to their frequency
- Plot # of solutions vs Frequency
  - (200,250): 250 solutions appeared 200 times each
- In theory, we expect a Poisson distribution
Uniformity Comparison

#Solutions vs. Frequency
Uniformity Comparison

#Solutions

Frequency

- Ideal Sampler
- UniGen2
- Parallel UniGen2
Outline

- Losing Independence of hashing functions
- Losing Independence among samples
- Parallelization of Constrained Random Simulation
- Conclusion
### How well did we tradeoff Independence?

<table>
<thead>
<tr>
<th>Relaxation Independence</th>
<th>Loss</th>
<th>Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hashing</td>
<td>Uniformity to Almost Uniformity</td>
<td>• 2-3 orders of magnitude performance improvement</td>
</tr>
</tbody>
</table>
| Sample                  | Weakened Almost Uniformity | • Still provides coverage guarantees  
|                         |                        | • 20 x improvement  
|                         |                        | • Parallelization  
|                         |                        | • Achieved desired performance |

- **Relaxation Independence**: Independence is relaxed in order to achieve performance gains.
- **Loss**: Uniformity is weakened to almost uniformity.
- **Gain**: 2-3 orders of magnitude performance improvement.
Takeaways

▪ Uniform generation has diverse applications

▪ Proposed the first scalable parallel approach that provides strong guarantees

▪ Requires \textit{polynomial} constant number of SAT calls per sample

▪ Scales linearly with number of cores

▪ Achieves desired performance by EDA Industry
New Paradigm of Simulation-based Verification
And one more thing!

- Tool (along with source code) is available online:
  
  http://tinyurl.com/unigen2

- Visit www.kuldeepmeel.com for papers/reports