Designing Scalable Techniques for Dynamic Verification and Probabilistic Inference

Kuldeep S. Meel
Rice University

Joint work with Alexander Ivrii (IBM), Supratik Chakraborty (IITB), Daniel J. Fremont(UCB), Sharad Malik (Princeton), Sanjit A. Seshia (UCB), Moshe Y. Vardi (Rice)
How do we guarantee that systems work correctly?

Functional Verification

- Formal verification
  - Challenges: formal requirements, scalability
  - 10-15% of verification effort (my estimate)
- Dynamic verification: dominant approach
Dynamic Verification

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

How do we verify that circuit works?

- Try for all values of $a$ and $b$
  - $2^{128}$ possibilities
  - Sun will go nova before done!
  - Not scalable

$c = f(a, b)$
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
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. $232 \times_{32} a + b \neq 1100$
  2. $1020 <_{64} (b /_{64} 2) +_{64} a <_{64} 2200$

- Test vectors: solutions of constraints
  - Proposed by Lichtenstein, Malka, Aharon (IAAI 94)
Constraint satisfaction for random stimuli generation

Yehuda Naveh
IBM Haifa Research Lab
Constrained-Random Simulation

Problem: How can we uniformly sample the values of a and b satisfying the above constraints?

Sources for Constraints

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

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

- Users:
  1. \(^{32}232 \times_{32}^{a} + b \neq 1100\)
  2. \(^{64}1020 < {_{64}^{(b /_{64}^{2})}} + {_{64}^{a}} < {_{64}^{2200}}\)
Problem Formulation

Given a SAT formula, sample solutions uniformly, while scaling to real world problems.

Set of Constraints

SAT Formula

Scalable Uniform Generation of SAT-Witnesses
Diverse Applications

- Search-based Synthesis
- Probabilistic Inference
- Planning under uncertainty
- Automatic Problem Generation
- Constrained Random Simulation
Search-Based Synthesis

- **Goal**: synthesize from under-constrained specifications ("sketch")
- Large space of programs that satisfy correctness conditions
- Task: Find “optimal” program (wrt running time, memory, ...)
- Method: *Uniformly sample* from the space of programs
Outline

- Sampling Techniques for Dynamic Verification
- Extension to approximate probabilistic inference
- Construction of Efficient Hashing functions
- Future Directions
Uniform Generation

Ref: “A Scalable Near-Uniform Generator” (CAV 2013)
“Balancing Scalability and Uniformity in SAT-Witness Generator” (DAC 2014)
“On Parallel Scalable Generation of SAT-Witnesses” (TACAS 2015)
## Prior Work

<table>
<thead>
<tr>
<th>BDD-based</th>
<th>SAT-based heuristics</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Guarantees:</strong> strong</td>
<td><strong>Guarantees:</strong> weak</td>
</tr>
<tr>
<td><strong>Performance:</strong> weak</td>
<td><strong>Performance:</strong> strong</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Theoretical Work</th>
<th>Heuristic Work</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Guarantees:</strong> strong</td>
<td><strong>Guarantees:</strong> weak</td>
</tr>
<tr>
<td><strong>Performance:</strong> weak</td>
<td><strong>Performance:</strong> strong</td>
</tr>
</tbody>
</table>

- **BGP Algorithm**
- **XORSample**
Our Contribution

- **BDD-based**
  - **Guarantees:** strong
  - **Performance:** weak

- **Theoretical Work**
  - **Guarantees:** strong
  - **Performance:** weak

- **UniGen**
  - **Guarantees:** strong
  - **Performance:** strong

- **Heuristic Work**
  - **Guarantees:** weak
  - **Performance:** strong

- **BGP Algorithm**

- **SAT-based heuristics**
  - **Guarantees:** weak
  - **Performance:** strong

- **XORSample**

INDUSTRY

ACADEMIA
Partitioning into equal “small” cells
Partitioning into equal “small” cells
Partitioning into equal “small” cells

Pick a random cell

Pick a random solution from this cell
How to Partition?

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

Universal Hashing
[Carter-Wegman 1979] (IBM Research)
Universal Hashing

- Hash functions: mapping \(\{0,1\}^n\) to \(\{0,1\}^m\)
  - \(2^n\) elements to \(2^m\) 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)
Universal Hashing and Independence

- Hash functions from mapping \(\{0,1\}^n\) to \(\{0,1\}^m\)
  - \((2^n \text{ elements to } 2^m \text{ cells})\)

- Universal hash functions:
  - Choose hash function randomly
  - For arbitrary distribution on inputs => All cells are *roughly* equal in expectation

- But:
  - While each input is hashed *uniformly*
  - Different inputs *might not* be hashed *independently*
Strong Universality

- $H(n,m,r)$: Family of $r$-universal hash functions mapping \{0,1\}^n to \{0,1\}^m ($2^n$ elements to $2^m$ cells)
  - $r$: degree of independence of hashed inputs

- Higher $r$ => Stronger guarantee on range of size of cells

- $r$-wise universality => Polynomials of degree $r-1$

- Higher universality => Higher complexity
Hashing-based Approaches

Solution space

n-universal hashing

BGP Algorithm

All cells are small

Uniform Generation
Scaling to Thousands of Variables

n-universal hashing

3-universal hashing

Solution space

Random

BGP Algorithm

All cells are small

Uniform Generation

UniGen

Only a randomly chosen cell needs to be “small”

Almost-Uniform Generation
Scaling to 100K Variables

Solution space

From tens of variables to 100K variables!

BGP Algorithm

UniGen

All cells should be small

Only a randomly chosen cells needs to be “small”

Uniform Generation

Almost-Uniform Generation
Notions of Uniformity

- **Uniformity**

  For every solution $y$ of $R_F$

  $$\Pr[y \text{ is output}] = \frac{1}{|R_F|}$$

- **Almost-Uniformity**

  For every solution $y$ of $R_F$

  $$\frac{1}{(1+\varepsilon)} \times \frac{1}{|R_F|} \leq \Pr[y \text{ is output}] \leq \frac{(1+\varepsilon)}{|R_F|}$$
Partitioning

- How large should the cells be?

- How many cells?
Size of cell

- Too large => Hard to enumerate
- Too small => Variance can be very high

\[ \text{pivot} = 5(1 + 1/\varepsilon)^2 \]
UniGen

\[ R_F \]

# of sols < pivot?

NO
UniGen

# of sols < pivot?

YES
UniGen

Select a solution randomly from cell.

Non-empty

Empty

FAIL

# of sols < pivot
Strong Theoretical Guarantees

- **Almost-Uniformity**

  For every solution $y$ of $R_F$

  \[
  \frac{1}{(6.84+\varepsilon) \times 1/|R_F|} \leq \Pr[y \text{ is output}] \leq \frac{(6.84+\varepsilon)}{|R_F|}
  \]

- **Success Probability**

  UniGen succeeds with probability at least 0.52

  - In practice, succ. Probability $\sim 0.99$

- **Polynomial number of calls to SAT Solver**
Results: Uniformity

- Benchmark: case110.cnf; \#var: 287; \#clauses: 1263
- Total Runs: $4 \times 10^6$; Total Solutions: 16384

![Graph showing frequency of solutions with two lines representing US and UniGen]
2-3 Orders of Magnitude Faster

![Bar chart showing time comparisons between UniGen and XORSample'] for various benchmarks. The y-axis represents time in seconds, ranging from 0.1 to 1000000, and the x-axis lists different benchmarks like `case47`, `case105`, `case8`, etc. The chart illustrates that UniGen performs significantly faster than XORSample' for most benchmarks.
Outline

- Sampling Techniques for Dynamic Verification

- Extension to approximate probabilistic inference

- Construction of Efficient Hashing functions

- Future Directions
Extension to Approximate Probabilistic Inference

Ref: “A Scalable Approximate Model Counter” (CP 2013)
Probabilistic Inference

How do we infer useful information from the data filled with uncertainty?

Modeling Attendance for Today’s Talk

Pr(Attending Talk | Interest in topic = True)

Roth, 1996
Model Counting

- Model Counting: Given a Boolean Formula $F$, count the number of models of $F$.

\[
F = (a \lor b)
\]
\[
R_F := \{(a = 0, b = 1), (a = 1, b = 0), (a = 1, b = 1)\}
\]
\[
|R_F| = 3
\]

- \#P-complete
  - \#P: Class of counting problem whose decision problems lie in NP
Practical Applications

Wide range of applications!

- Estimating coverage achieved
- Probabilistic reasoning/Bayesian inference
- Planning with uncertainty
- Multi-agent/ adversarial reasoning

[Roth 96, Sang 04, Bacchus 04, Domshlak 07]
Counting through Partitioning
Counting through Partitioning

Pick a random cell

Total # of solutions = # solutions in the cell * total # of cells
ApproxMC in Action
ApproxMC in Action

Median

Algorithm

690  710  730  730  731  831  ............  834

t
Strong Theoretical Results

ApproxMC(F, tolerance: e, confidence parameter: d)
Suppose ApproxMC(F, , ) returns C. Then

$$\Pr\left[ \frac{|R_F|}{1+|R_F|(1+\epsilon)} \leq C \right] \geq 1$$

ApproxMC runs in time polynomial in $F, 1, \log(1-d)$ relative to SAT oracle
Results: Performance Comparison

[Graph showing performance comparison between ApproxMC and Cachet over time (seconds)]
Results: Performance Comparison

ApproxMC

Cachet
Can Solve a Large Class of Problems

Large class of problems that lie beyond the exact counters but can be computed by ApproxMC
Mean Error: Only 4% (allowed: 75%)

Mean error: 4% – much smaller than the theoretical guarantee of 75%
Approximate Weighted Counting

Weighted Counting

**Given**
- CNF Formula $F$
- Weight Function $W$ over assignments

**Problem**
- What is the sum of weights of *satisfying* assignments?

**Example**
- $F = (a \lor b)$
- $W([0,1]) = W([1,0]) = 1/3 \quad W([1,1]) = W([0,0]) = 1/6$
- $W(F) = 1/3 + 1/3 + 1/6 = 5/6$
Partition into (weighted) equal “small” cells
Partition into (weighted) equal “small” cells

Pick a random cell

Weighted Count = Weight of random cell * total # of cells
Can you always achieve partitioning?

What if one solution dominates the entire solution space

Tilt = $w_{\text{max}} / w_{\text{min}}$

Small tilt $\rightarrow$ All solutions contribute
How to handle large tilt?

Tilt = 992
Handling Large Tilt

Can be achieved with Pseudo-Boolean Solver
Still a SAT problem not Optimization
Outline

▪ Sampling Techniques for Dynamic Verification

▪ Extension to approximate probabilistic inference

▪ Construction of Efficient Hashing functions

▪ Future Directions
Construction of Efficient Hash Functions


Best Student Paper Award
XOR-Based Hashing

- 3-universal hashing
- Partition $2^n$ space into $2^m$ cells
- Variables: $X_1, X_2, X_3, \ldots, X_n$
- Pick every variable with prob. $\frac{1}{2}$, XOR them and equate to 0/1 with prob. $\frac{1}{2}$
- $X_1 + X_3 + X_6 + \ldots + X_{n-1} = 0$ (splits solution space)
- $m$ XOR equations -> $2^m$ cells
- The cell: $F$ XOR (CNF+XOR)
XOR-Based Hashing

- **CryptoMiniSAT**: Efficient for CNF+XOR
- Avg Length : \( n/2 \)
- Smaller the XORs, better the performance

How to shorten XOR clauses?
Independent Support

- Set of variables such that assignments to these uniquely determine assignments to rest of variables for formula to be true
- $c \leftrightarrow (a \lor b)$; Independent Support (I): \{a, b\}
- If $s_1$ and $s_2$ agree on I then $s_1 = s_2$
- Hash only on the independent variables
Computing Minimal Independent Support

- Reduction to the problem of computing MUS (Minimal Unsatisfiable Subset)

- Minimal Independent supports are $\frac{1}{100} – \frac{1}{1000}$ of the size of $X$

- Provides 1-2 orders of magnitude
Future Directions
Extension to More Expressive Domains (SMT, CSP)

- Efficient 3-independent hashing schemes
  - Extending bit-wise XOR to SMT provides guarantees but no advantage of SMT progress

- Solvers to handle F + Hash efficiently
  - CryptoMiniSAT has fueled progress for SAT domain
  - Similar solvers for other domains?
Handling Distributions

- Design of Pseudo-Boolean solvers to handle tilt
- Classification of problems according to tilt
- Online estimation of tilt
- Other techniques for high-tilt distributions
Questions?

Papers and tools: http://www.kuldeepmeel.com