NP? No Problem! An Invitation to the World of Formal Methods

Kuldeep S. Meel

School of Computing

National University of Singapore

Ph.D. (2017), advised by Prof. Supratik Chakraborty (IIT Bombay) and Prof. Moshe Vardi (Rice University)
The Quest for Automated Formal Reasoning

All Greeks are humans
All humans are mortal
All Greeks are mortal

Assign Symbols & Use Algebra

\[
\begin{align*}
g & \rightarrow h \\
h & \rightarrow m \\
g & \rightarrow m
\end{align*}
\]
The Quest for Automated Formal Reasoning

All Greeks are humans
All humans are mortal
All Greeks are mortal

Assign Symbols & Use Algebra

Central Equation: Is it always the case that \(((g \rightarrow h) \land (h \rightarrow m)) \rightarrow (g \rightarrow m)\)?
Or Equivalently, can it be the case \(((g \rightarrow h) \land (h \rightarrow m)) \rightarrow !(g \rightarrow m)\)?
The Quest for Automated Formal Reasoning

All Greeks are humans
All humans are mortal
All Greeks are mortal

Assign Symbols & Use Algebra

Central Equation: Is it always the case that \(((g \rightarrow h) \land (h \rightarrow m)) \rightarrow (g \rightarrow m)\)?
Or Equivalently, can it be the case \(((g \rightarrow h) \land (h \rightarrow m)) \rightarrow ! (g \rightarrow m)\)?

William Stanley Jevons, 1835-1882: “I have given much attention, therefore, to lessening both the manual and mental labour of the process, and I shall describe several devices which may be adopted for saving trouble and risk of mistake.”
Boolean Satisfiability (SAT): Given a Boolean expression, using “and” (∧) “or” (∨) and “not” (¬), is there a satisfying solution (an assignment of 0’s and 1’s to the variables that makes the expression equal 1)?

Example:

\[ (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_2 \lor \neg x_3 \lor x_4) \land (x_3 \lor x_1 \lor x_4) \]

Solution: \( x_1 = 0, \ x_2 = 0, \ x_3 = 1, \ x_4 = 1 \)
Boolean Satisfiability (SAT): Given a Boolean expression, using “and” (\( \land \)) “or”, (\( \lor \)) and “not” (\( \neg \)), is there a satisfying solution (an assignment of 0’s and 1’s to the variables that makes the expression equal 1)?

Example:

\[
(\neg x_1 \lor x_2 \lor x_3) \land (\neg x_2 \lor \neg x_3 \lor x_4) \land (x_3 \lor x_1 \lor x_4)
\]

Solution: \( x_1 = 0, \ x_2 = 0, \ x_3 = 1, \ x_4 = 1 \)

- Ernst Schröder, 1841-1902: “Getting a handle on the consequences of any premises, or at least the fastest method for obtaining these consequences, seems to me to be one of the noblest, if not the ultimate goal of mathematics and logic.”
Algorithmic Boolean Reasoning: Early History

- **Davis and Putnam, 1958**: “Computational Methods in The Propositional calculus”, unpublished report to the NSA
- **Davis and Putnam, JACM 1960**: “A Computing procedure for quantification theory”
- **Davis, Logemann, and Loveland, CACM 1962**: “A machine program for theorem proving”
The DPLL algorithm
The DPLL algorithm

\[ \mathcal{F} = (x \lor y) \land (a \lor b) \land (\bar{a} \lor b) \land (a \lor \bar{b}) \land (\bar{a} \lor \bar{b}) \]

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>\emptyset</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>\text{x}</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>\text{y}</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>\text{a}</td>
<td></td>
</tr>
</tbody>
</table>
The DPLL algorithm

\[ \mathcal{F} = (x \lor y) \land (a \lor b) \land (\bar{a} \lor b) \land (a \lor \bar{b}) \land (\bar{a} \lor \bar{b}) \]

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( \emptyset )</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>( x )</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>( y )</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>( a \rightarrow b \rightarrow \bot )</td>
<td></td>
</tr>
</tbody>
</table>
The DPLL algorithm

\[ \mathcal{F} = (x \lor y) \land (a \lor b) \land (\bar{a} \lor b) \land (a \lor \bar{b}) \land (\bar{a} \lor \bar{b}) \]

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>∅</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>y</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>\bar{a}</td>
<td>\bar{b} \perp</td>
</tr>
</tbody>
</table>

Slide 5/26
The DPLL algorithm

\( \mathcal{F} = (x \lor y) \land (a \lor b) \land (\overline{a} \lor b) \land (a \lor \overline{b}) \land (\overline{a} \lor \overline{b}) \)

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>\emptyset</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>(x)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>(\overline{y})</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>(a)</td>
<td>(b)</td>
</tr>
</tbody>
</table>

Slide 5/26
The DPLL algorithm

\[ \mathcal{F} = (x \lor y) \land (a \lor b) \land (\bar{a} \lor b) \land (a \lor \bar{b}) \land (\bar{a} \lor \bar{b}) \]

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>\emptyset</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>\ x</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>\ \bar{y}</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>\ \bar{a}</td>
<td>\ \rightarrow \ \bar{b}</td>
</tr>
</tbody>
</table>

Slide 5/26
The DPLL algorithm

\[ \mathcal{F} = (x \lor y) \land (a \lor b) \land (\bar{a} \lor b) \land (a \lor \bar{b}) \land (\bar{a} \lor \bar{b}) \]

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>\emptyset</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>\bar{x}</td>
<td>y</td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td>b</td>
</tr>
</tbody>
</table>

Slide 5/26
The DPLL algorithm

\[ \mathcal{F} = (x \lor y) \land (a \lor b) \land (\bar{a} \lor b) \land (a \lor \bar{b}) \land (\bar{a} \lor \bar{b}) \]

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>\emptyset</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>\bar{x}</td>
<td>y</td>
</tr>
<tr>
<td>2</td>
<td>\bar{a}</td>
<td>\bar{b} \rightarrow \bot</td>
</tr>
</tbody>
</table>

Diagram:

Slide 5/26
The Dreaded 70s

Hoare, 1969: Proving correctness of the programs can be reduced to theorem proving
  • For a large interesting class of programs, proving correctness reduces to SAT
The Dreaded 70s

Hoare, 1969: Proving correctness of the programs can be reduced to theorem proving
  • For a large interesting class of programs, proving correctness reduces to SAT

Cook, 1971, Levin, 1972: SAT is NP-complete
The Dreaded 70s

Hoare, 1969: Proving correctness of the programs can be reduced to theorem proving
  • For a large interesting class of programs, proving correctness reduces to SAT

Cook, 1971, Levin, 1972: SAT is NP-complete

Vardi: “When I was a graduate student in 1970’s, SAT was a scary problem, not to be touched by a 10 foot pole”
The Dreaded 70s

Hoare, 1969: Proving correctness of the programs can be reduced to theorem proving
  • For a large interesting class of programs, proving correctness reduces to SAT

Cook, 1971, Levin, 1972: SAT is NP-complete

Vardi: “When I was a graduate student in 1970’s, SAT was a scary problem, not to be touched by a 10 foot pole”

(Cartoon adapted from Gary Johnson)

De Millo, Lipton, Perlis, 1979: “formal verifications of programs, no matter how obtained, will not play the same key role in the development of computer science and software engineering as proofs do in mathematics.”
The Cautious 80’s followed by the storm

Clark, Emerson, 1981: “We argue that proof construction is unnecessary in the case of finite state concurrent systems and can be replaced by a model-theoretic approach which will mechanically determine if the system meets a specification expressed in propositional temporal logic”

Burch, EM Clarke, KL McMillan, DL Dill, LJ Hwang, 1992: Symbolic model checking, restricted to Binary Decision Diagrams: formulas for which Satisfiability is PTIME
Clark, Emerson, 1981: “We argue that proof construction is unnecessary in the case of finite state concurrent systems and can be replaced by a model-theoretic approach which will mechanically determine if the system meets a specification expressed in propositional temporal logic”

Burch, EM Clarke, KL McMillan, DL Dill, LJ Hwang, 1992: Symbolic model checking, restricted to Binary Decision Diagrams: formulas for which Satisfiability is PTIME

Pentium FDIV Bug, 1994: The recall due to floating point unit bug cost $500 million

Ariane 5, 1996: The rocket exploded only after 40 seconds due to exception handling
The Cautious 80’s followed by the storm

Clark, Emerson, 1981: “We argue that proof construction is unnecessary in the case of finite state concurrent systems and can be replaced by a model-theoretic approach which will mechanically determine if the system meets a specification expressed in propositional temporal logic”

Burch, EM Clarke, KL McMillan, DL Dill, LJ Hwang , 1992: Symbolic model checking, restricted to Binary Decision Diagrams: formulas for which Satisfiability is PTIME

Pentium FDIV Bug, 1994: The recall due to floating point unit bug cost $500 million

Ariane 5, 1996: The rocket exploded only after 40 seconds due to exception handling

Marques-Silva and Sakallah, 1996: “GRASP is premised on the inevitability of conflicts during the search and its most distinguishing feature is the augmentation of basic backtracking search with a powerful conflict analysis procedure”
Clause learning

\[(\overline{a} \lor \overline{b}) \land (\overline{z} \lor b) \land (\overline{x} \lor \overline{z} \lor a) \land (y \lor b)\]

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ϕ</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>y</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>z</td>
<td>a, √, ⊥</td>
</tr>
</tbody>
</table>

- Analyze conflict
- \[\text{MSS96a, MSS96b, MSS96c, MSS96d, MSS99}\]
- Reasons:
  - Decision variable & literals assigned at decision levels less than current
- Create new clause: \[(\overline{x} \lor \overline{a})\]
- Can relate clause learning with resolution
  - Learned clauses result from \((\text{selected})\) resolution operations

Slide 8/26
Clause learning

$$(\overline{a} \lor \overline{b}) \land (\overline{z} \lor b) \land (\overline{x} \lor \overline{z} \lor a) \land (y \lor b)$$

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$\emptyset$</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>$x$</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>$y$</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>$z$</td>
<td>$a$</td>
</tr>
</tbody>
</table>

$\Rightarrow$ Analyze conflict

$[[MSS96a,MSS96b,MSS96c,MSS96d,MSS99]]$
Clause learning

\[(\overline{a} \lor \overline{b}) \land (\overline{z} \lor b) \land (\overline{x} \lor \overline{z} \lor a) \land (y \lor b)\]

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$\emptyset$</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>$x$</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>$y$</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>$z$ $\rightarrow$ $a \rightarrow \bot$ $b$</td>
<td></td>
</tr>
</tbody>
</table>

- Analyze conflict
  - Reasons: $x$ and $z$
    - Decision variable & literals assigned at decision levels less than current

[MSS96a,MSS96b,MSS96c,MSS96d,MSS99]
Clause learning

$$(\bar{a} \lor \bar{b}) \land (\bar{z} \lor b) \land (\bar{x} \lor \bar{z} \lor a) \land (y \lor b)$$

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$\emptyset$</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>$\times$</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>$y$</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>$z \rightarrow a \rightarrow b \rightarrow \bot$</td>
<td></td>
</tr>
</tbody>
</table>

- Analyze conflict
  - Reasons: $x$ and $z$
    - Decision variable & literals assigned at decision levels less than current
  - Create **new** clause: $(\bar{x} \lor \bar{z})$

[MSS96a,MSS96b,MSS96c,MSS96d,MSS99]
Clause learning

\[(\bar{a} \lor \bar{b}) \land (\bar{z} \lor b) \land (\bar{x} \lor \bar{z} \lor a) \land (y \lor b)\]

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>x</td>
<td>(\bar{a} \lor \bar{b})</td>
</tr>
<tr>
<td>2</td>
<td>y</td>
<td>(\bar{z} \lor b)</td>
</tr>
<tr>
<td>3</td>
<td>z</td>
<td>(\bar{x} \lor \bar{z} \lor a)</td>
</tr>
<tr>
<td></td>
<td>a</td>
<td></td>
</tr>
<tr>
<td></td>
<td>b</td>
<td></td>
</tr>
<tr>
<td></td>
<td>⊥</td>
<td></td>
</tr>
</tbody>
</table>

▶ Analyze conflict

▶ Reasons: \( x \) and \( z \)
  
  • Decision variable & literals assigned at decision levels less than current

▶ Create new clause: \( (\bar{x} \lor \bar{z}) \)

▶ Can relate clause learning with resolution

[MSS96a, MSS96b, MSS96c, MSS96d, MSS99]
Clause learning

\[(\overline{a} \lor \overline{b}) \land (\overline{z} \lor b) \land (\overline{x} \lor \overline{z} \lor a) \land (y \lor b)\]

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>∅</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>y</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>z</td>
<td>a</td>
</tr>
</tbody>
</table>

▶ Analyze conflict
▶ Reasons: \(x\) and \(z\)
  • Decision variable & literals assigned at decision levels less than current
▶ Create **new** clause: \((\overline{x} \lor \overline{z})\)
▶ Can relate **clause learning** with resolution

[MSS96a,MSS96b,MSS96c,MSS96d,MSS99]
Clause learning

\[(\bar{a} \lor \bar{b}) \land (\bar{z} \lor b) \land (\bar{x} \lor \bar{z} \lor a) \land (y \lor b)\]

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>\emptyset</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>(x)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>(y)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>(z) \rightarrow (a) \rightarrow \bot</td>
<td>(b)</td>
</tr>
</tbody>
</table>

- Analyze conflict
  - Reasons: \(x\) and \(z\)
    - Decision variable & literals assigned at decision levels less than current
  - Create new clause: \((\bar{x} \lor \bar{z})\)
- Can relate clause learning with resolution

\[\text{[MSS96a, MSS96b, MSS96c, MSS96d, MSS99]}\]
Clause learning

$$(\bar{a} \lor \bar{b}) \land (\bar{z} \lor b) \land (\bar{x} \lor \bar{z} \lor a) \land (y \lor b)$$

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$\emptyset$</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>$\times$</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>$y$</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>$z$</td>
<td>$a$</td>
</tr>
</tbody>
</table>

- Analyze conflict
  - Reasons: $x$ and $z$
    - Decision variable $x$ & literals assigned at decision levels less than current
  - Create new clause: $(\bar{x} \lor \bar{z})$
- Can relate clause learning with resolution
  - Learned clauses result from (selected) resolution operations

$$(\bar{a} \lor \bar{b})$$ $$(\bar{z} \lor b)$$ $$(\bar{x} \lor \bar{z} \lor a)$$

$$(\bar{a} \lor \bar{z})$$ $$(\bar{z} \lor b)$$ $$(\bar{x} \lor \bar{z} \lor a)$$

[MSS96a,MSS96b,MSS96c,MSS96d,MSS99]
Clause learning – after backtracking

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>∅</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>y</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>z</td>
<td>a</td>
</tr>
</tbody>
</table>

Backtracking differs from plain DPLL:

• Always backtrack after a conflict
Clause learning – after backtracking

Clause \((\overline{\bar{x}} \lor \overline{\bar{z}})\) is asserting at decision level 1
Clause learning – after backtracking

Clause \((\overline{x} \lor \overline{z})\) is asserting at decision level 1
Clause learning – after backtracking

Clause $(\overline{x} \lor \overline{z})$ is asserting at decision level 1

- Backtracking differs from plain DPLL:
  - Always backtrack after a conflict
The Roaring 2000s

- Clause learning & non-chronological backtracking
  - Exploit UIPs
  - Minimize learned clauses
  - Opportunistically delete clauses

- Search restarts
The Roaring 2000s

- Clause learning & non-chronological backtracking
  - Exploit UIPs
  - Minimize learned clauses
  - Opportunistically delete clauses

- Search restarts

- Lazy data structures
  - Watched literals

- Conflict-guided branching
  - Lightweight branching heuristics
  - Phase saving
The Roaring 2000s

- **Clause learning & non-chronological backtracking**
  - Exploit UIPs
  - Minimize learned clauses
  - Opportunistically delete clauses

- **Search restarts**

- **Lazy data structures**
  - Watched literals

- **Conflict-guided branching**
  - Lightweight branching heuristics
  - Phase saving

---

**Biere, Cimatti, Clarke, Zhu, 1999:** “We show how boolean decision procedures can replace BDDs. This new technique avoids the space blow up of BDDs, generates counterexamples much faster.”

---

**Clark, Emerson, Sifakis, 2007:** Turing Award

**Knuth, 2010s:** SAT is far from an abstract exercise in understanding formal systems. These so-called “SAT solvers” can now routinely find solutions to practical problems that involve millions of variables and were thought until very recently to be hopelessly difficult.
Beyond NP

[Circa 2012 @IIT Bombay]: Now that NP is “No Problem”, it is time to look beyond satisfiability

```cpp
PC2 (char[] SP, char[] UI) {
    match = true;
    for (int i=0; i<UI.length(); i++) {
        if (SP[i] != UI[i]) match=false;
        else match = match;
    }
    if (match) return Yes;
    else return No;
}
```

Information Leakage

Fairness

Robustness

Critical Infrastructure
Beyond NP

[Circa 2012 @IIT Bombay]: Now that NP is “No Problem”, it is time to look beyond satisfiability.

```c
PC2 (char[] SP, char[] UI) {
    match = true;
    for (int i=0; i<UI.length(); i++) {
        if (SP[i] != UI[i]) match = false;
        else match = match;
    }
    if (match) return Yes;
    else return No;
}
```

Information Leakage

Fairness

Robustness

Critical Infrastructure

Quantification: How often does $M$ satisfy $P$?

Counting
Can we predict the likelihood of a blackout due to natural disaster?
Resilience of Critical Infrastructure Networks

Can we predict the likelihood of a blackout due to natural disaster?

- \( G = (V, E) \); set of source nodes \( S \) and terminal node \( t \)
- failure probability \( g : E \to [0, 1] \)
- Compute \( \Pr[ t \text{ is disconnected from } S] \)?
Can we predict the likelihood of a blackout due to natural disaster?

- \( G = (V, E) \); set of source nodes \( S \) and terminal node \( t \)
- failure probability \( g : E \rightarrow [0, 1] \)
- Compute \( \Pr[ t \text{ is disconnected from } S] \)?

**Key Idea:** Encode disconnectedness using constraints

**Impact:** The first theoretically sound estimates of resilience in power transmission networks of ten medium sized cities in US
Robustness Quantification

\[ \{ x : \mathcal{N}(x + \varepsilon) \neq \mathcal{N}(x) \} \]
Robustness Quantification

\[ \{ x : N(x + \varepsilon) \neq N(x) \} \]

Encode Symbolically

Counting
Robustness Quantification

\[ \{ x : \mathcal{N}(x + \varepsilon) \neq \mathcal{N}(x) \} \]

Encode Symbolically

Fairness Quantification

\[ \{ x : \mathcal{N}(x \wedge \text{BLACK}) \neq \mathcal{N}(x \wedge \text{WHITE}) \} \]

Encode Symbolically

Counting
Quantitative Analysis of AI Systems

Robustness Quantification

\[ \{ x \in \mathcal{N}(x + \varepsilon) \neq \mathcal{N}(x) \} \]

Encode Symbolically

Counting

Impact: The first scalable technique for rigorous quantification of robustness and fairness of Binarized Neural Networks

Fairness Quantification

\[ \{ x \in \mathcal{N}(x \land \text{BLACK}) \neq \mathcal{N}(x \land \text{WHITE}) \} \]

Encode Symbolically
Counting

- **Given**: A Boolean formula $F$ over $X_1, X_2, \cdots X_n$
- **$\text{Sol}(F)$**: $\{\text{solutions of } F\}$
- **$\text{SAT}$**: Determine if $\text{Sol}(F)$ is non-empty
- **$\text{Counting}$**: Determine $|\text{Sol}(F)|$

Example:

$F := (X_1 \lor X_2)$

- $\text{Sol}(F) = \{(0, 1), (1, 0), (1, 1)\}$

- $|\text{Sol}(F)| = 3$

Valiant, 1979: Counting exactly is #P-hard

Stockmeyer, 1983: Probably Approximately Correct (PAC) aka $(\epsilon, \delta)$-guarantees

$$\Pr |\text{Sol}(F)| \leq 1 + \epsilon \leq \text{ApproxCount}(F, \epsilon, \delta) \leq (1 + \epsilon) |\text{Sol}(F)| \geq 1 - \delta$$
Counting

- **Given**: A Boolean formula $F$ over $X_1, X_2, \cdots X_n$
- **Sol($F$)** = \{ solutions of $F$ \}
- **SAT**: Determine if Sol($F$) is non-empty
- **Counting**: Determine $|\text{Sol}(F)|$
- **Example**: $F := (X_1 \lor X_2)$
  - **Sol($F$)** = \{(0, 1), (1, 0), (1, 1)\}
  - $|\text{Sol}(F)| = 3$

Valiant, 1979: Counting exactly is $\#P$-hard
Counting

- **Given**: A Boolean formula $F$ over $X_1, X_2, \ldots, X_n$
- **Sol($F$)** = \{ solutions of $F$ \}
- **SAT**: Determine if Sol($F$) is non-empty
- **Counting**: Determine $|\text{Sol}($F$)|$
- **Example**: $F := (X_1 \lor X_2)$
  - Sol($F$) = \{(0, 1), (1, 0), (1, 1)\}
  - $|\text{Sol}($F$)| = 3$

Valiant, 1979: Counting exactly is **#P-hard**

Stockmeyer, 1983: Probably Approximately Correct (PAC) aka $(\varepsilon, \delta)$-guarantees

$$
\text{Pr} \left[ \frac{|\text{Sol}($F$)|}{1 + \varepsilon} \leq \text{ApproxCount}(F, \varepsilon, \delta) \leq (1 + \varepsilon)|\text{Sol}($F$)| \right] \geq 1 - \delta
$$
Snapshot from 2012

State of the art tool in 2012 could handle one out of 10^76 robustness instances.

Can we bridge the gap between theory and practice?
Snapshot from 2012

State of the art tool in 2012 could handle one out of 10^76 robustness instances.

Can we bridge the gap between theory and practice?
State of the art tool in 2012 could handle one out of 1076 robustness instances

Can we bridge the gap between theory and practice?
State of the art tool in 2012 could handle one out of 1076 robustness instances

Can we bridge the gap between theory and practice?
Counting in Mumbai

How many people in Mumbai like coffee?

- Population of Mumbai = 12.5M
- Assign every person a unique \( (n =) \) 24 bit identifier \( (2^n \approx 12.5M) \)
Counting in Mumbai

How many people in Mumbai like coffee?

- Population of Mumbai = 12.5M
- Assign every person a unique \( n \) 24 bit identifier \( 2^n \approx 12.5M \)

- Attempt #1: Pick 50 people and count how many of them like coffee and multiply by 12.5M/50
Counting in Mumbai

How many people in Mumbai like coffee?

- Population of Mumbai = 12.5M
- Assign every person a unique \( (n =) \) 24 bit identifier \( (2^n \approx 12.5M) \)

- Attempt #1: Pick 50 people and count how many of them like coffee and multiply by 12.5M/50
  - If only 1000 people like coffee, it is unlikely that we will find anyone who likes coffee in our sample of 50
Counting in Mumbai

How many people in Mumbai like coffee?

- Population of Mumbai = 12.5M
- Assign every person a unique \( n = 24 \) bit identifier \( 2^n \approx 12.5M \)

- Attempt #1: Pick 50 people and count how many of them like coffee and multiply by 12.5M/50
  - If only 1000 people like coffee, it is unlikely that we will find anyone who likes coffee in our sample of 50

- SAT Query: Find a person who likes coffee
Counting in Mumbai

How many people in Mumbai like coffee?

- Population of Mumbai = 12.5M
- Assign every person a unique \( n = 24 \) bit identifier \( 2^n \approx 12.5M \)

- Attempt #1: Pick 50 people and count how many of them like coffee and multiply by 12.5M/50
  - If only 1000 people like coffee, it is unlikely that we will find anyone who likes coffee in our sample of 50

- SAT Query: Find a person who likes coffee

- A SAT solver can answer queries like:
  - Q1: Find a person who likes coffee
  - Q2: Find a person who likes coffee and is not person \( y \)
Counting in Mumbai

How many people in Mumbai like coffee?

- Population of Mumbai = 12.5M
- Assign every person a unique \((n =)\) 24 bit identifier \((2^n \approx 12.5M)\)

- Attempt #1: Pick 50 people and count how many of them like coffee and multiply by 12.5M/50
  - If only 1000 people like coffee, it is unlikely that we will find anyone who likes coffee in our sample of 50

- SAT Query: Find a person who likes coffee
- A SAT solver can answer queries like:
  - Q1: Find a person who likes coffee
  - Q2: Find a person who likes coffee and is not person \(y\)

- Attempt #2: Enumerate every person who likes coffee
Counting in Mumbai

How many people in Mumbai like coffee?

- Population of Mumbai = 12.5M
- Assign every person a unique \( n = 24 \) bit identifier \( 2^n \approx 12.5M \)

- Attempt #1: Pick 50 people and count how many of them like coffee and multiply by 12.5M/50
  - If only 1000 people like coffee, it is unlikely that we will find anyone who likes coffee in our sample of 50

- SAT Query: Find a person who likes coffee
- A SAT solver can answer queries like:
  - Q1: Find a person who likes coffee
  - Q2: Find a person who likes coffee and is not person \( y \)

- Attempt #2: Enumerate every person who likes coffee
  - Potentially \( 2^n \) queries

Can we do with lesser # of SAT queries – \( O(n) \) or \( O(\log n) \)?
As Simple as Counting Dots

Pick a random cell

Estimate = Number of solutions in a cell × Number of cells
As Simple as Counting Dots

Pick a random cell

Estimate = Number of solutions in a cell \times Number of cells
As Simple as Counting Dots

Pick a random cell

Estimate = Number of solutions in a cell × Number of cells
Challenges

**Challenge 1** How to partition into *roughly equal small* cells of solutions without knowing the distribution of solutions?

**Challenge 2** How many cells?
Challenges

**Challenge 1** How to partition into *roughly equal small* cells of solutions without knowing the distribution of solutions?

- Designing function $h : \text{assignments} \rightarrow \text{cells}$ (hashing)

2-wise Independent Hashing \[\text{[CW77]}\]
Challenges

**Challenge 1** How to partition into *roughly equal small* cells of solutions without knowing the distribution of solutions?

- Designing function $h : \text{assignments} \rightarrow \text{cells}$ (hashing)
- Deterministic $h$ unlikely to work
**Challenge 1** How to partition into **roughly equal small** cells of solutions without knowing the distribution of solutions?

- Designing function $h : \text{assignments} \rightarrow \text{cells}$ (hashing)
- Deterministic $h$ unlikely to work
- Choose $h$ randomly from a large family $H$ of hash functions

2-wise Independent Hashing [CW77]
To construct $h : \{0, 1\}^n \rightarrow \{0, 1\}^m$, choose $m$ random XORs.

Pick every $X_i$ with prob. $\frac{1}{2}$ and XOR them.

$X_1 \oplus X_3 \oplus X_6 \cdots \oplus X_{n-2}$
• To construct $h : \{0, 1\}^n \to \{0, 1\}^m$, choose $m$ random XORs

• Pick every $X_i$ with prob. $\frac{1}{2}$ and XOR them
  \begin{align*}
  X_1 \oplus X_3 \oplus X_6 \cdots \oplus X_{n-2}
  \end{align*}

• To choose $\alpha \in \{0, 1\}^m$, set every XOR equation to 0 or 1 randomly

\begin{align*}
X_1 \oplus X_3 \oplus X_6 \cdots \oplus X_{n-2} &= 0 \quad (Q_1) \\
X_2 \oplus X_5 \oplus X_6 \cdots \oplus X_{n-1} &= 1 \quad (Q_2) \\
\cdots & \quad (\cdots) \\
X_1 \oplus X_2 \oplus X_5 \cdots \oplus X_{n-2} &= 1 \quad (Q_m)
\end{align*}

• Solutions in a cell: $F \land Q_1 \cdots \land Q_m$
Challenges

**Challenge 1** How to partition into *roughly equal small* cells of solutions without knowing the distribution of solutions?

Random XOR-based Hash Functions [CW77]

**Challenge 2** How many cells?
Challenge 2: How many cells?

- A cell is small if it has \( \approx \text{thresh} = 5\left(1 + \frac{1}{\epsilon}\right)^2 \) solutions
- Many solutions \( \implies \) Many cells & Fewer solutions \( \implies \) Fewer cells

\[ \text{Theorem: } \Pr_{\mathcal{H}}|\text{Sol}(F)| \leq 1 + \epsilon \leq \text{ApproxMC}(F, \epsilon, \delta) \leq |\text{Sol}(F)| \left(1 + \epsilon\right) \geq 1 - \delta \]

ApproxMC makes \( O\left(\frac{1}{\epsilon^2} \cdot \log \frac{1}{\delta} \cdot \log n\right) \) SAT queries.
Challenge 2: How many cells?

- A cell is small if it has $\approx \text{thresh} = 5(1 + \frac{1}{\epsilon})^2$ solutions
- Many solutions $\implies$ Many cells & Fewer solutions $\implies$ Fewer cells

Theorem: $\Pr_{|\text{Sol}(F)| \leq 1 + \epsilon} \leq \text{ApproxMC}(F, \epsilon, \delta) \leq |\text{Sol}(F)| (1 + \epsilon)$

ApproxMC makes $O\left(\frac{1}{\epsilon^2} \cdot \log \frac{1}{\delta} \cdot \log n\right)$ SAT queries.
Challenge 2: How many cells?

- A cell is small if it has $\approx \text{thresh} = 5(1 + \frac{1}{\varepsilon})^2$ solutions
- Many solutions $\implies$ Many cells & Fewer solutions $\implies$ Fewer cells

\[ \text{No} \]

\[ \# \text{ of sols} \leq \text{thresh} ? \]

\[ \# \text{ of sols} \leq \text{thresh} ? \]

\[ F \land Q_1 \]

[CMV13, CMV16]
Challenge 2: How many cells?

- A cell is small if it has \( \approx \text{thresh} = 5(1 + \frac{1}{\varepsilon})^2 \) solutions
- Many solutions \( \implies \) Many cells & Fewer solutions \( \implies \) Fewer cells

\[ \text{No} \]

\[ F \land Q_1 \leq \text{thresh?} \]

\[ F \land Q_1 \land Q_2 \]

[CMV13, CMV16]
Challenge 2: How many cells?

- A cell is small if it has $\approx \text{thresh} = 5(1 + \frac{1}{\varepsilon})^2$ solutions
- Many solutions $\Rightarrow$ Many cells & Fewer solutions $\Rightarrow$ Fewer cells

Theorem: $\Pr_{\text{Pr}}(\text{Sol}(F) \mid 1 + \varepsilon \leq \text{ApproxMC}(F, \varepsilon, \delta) \leq \text{Sol}(F)(1 + \varepsilon)^i \geq 1 - \delta$)

ApproxMC makes $O\left(\frac{1}{\varepsilon^2} \cdot \log \frac{1}{\delta} \cdot \log n\right)$ SAT queries.
Challenge 2: How many cells?

- A cell is small if it has $\approx \text{thresh} = 5(1 + \frac{1}{\varepsilon})^2$ solutions
- Many solutions $\implies$ Many cells & Fewer solutions $\implies$ Fewer cells

Estimate = $\# \text{ of sols} \times \# \text{ of cells}$
Challenge 2: How many cells?

- A cell is small if it has $\approx \text{thresh} = 5(1 + \frac{1}{\varepsilon})^2$ solutions
- Many solutions $\implies$ Many cells & Fewer solutions $\implies$ Fewer cells

Theorem: $\Pr \left[ \frac{|\text{Sol}(F)|}{1+\varepsilon} \leq \text{ApproxMC}(F, \varepsilon, \delta) \leq |\text{Sol}(F)|(1 + \varepsilon) \right] \geq 1 - \delta$

ApproxMC makes $O\left(\frac{1}{\varepsilon^2} \cdot \log \frac{1}{\delta} \cdot \log n\right)$ SAT queries.
Handle **reasonable** formulas: **reasonable** grids, **reasonable** programs
ApproxMC: Early Years (2013-17)

Handle **reasonable** formulas: **reasonable** grids, **reasonable** programs

**B. Cook**: Virtuous cycle: application areas drives more investment in foundational tools, while improvements in the foundational tools drive further applications. Around and around.
In Pursuit of Scalability (2017-now)

<table>
<thead>
<tr>
<th>Theoretical Advances</th>
<th>Algorithmic Engineering</th>
<th>Software Development</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sparse hashing</td>
<td>Duality</td>
<td>Phase Transition</td>
</tr>
<tr>
<td>SAT-20 LICS-20</td>
<td>CP-19</td>
<td>IJCAI-16,17,19 CP-20</td>
</tr>
<tr>
<td>Duality CP-19</td>
<td>CP-18, IJCAI-19 PODS-21,22</td>
<td></td>
</tr>
<tr>
<td>DNF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phase Transition IJCAI-16,17,19 CP-20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Indep Supp</td>
<td>Chain Formulas</td>
<td>Symmetry</td>
</tr>
<tr>
<td>Constraints-16 ICCAD-22</td>
<td>IJCAI-15 NeurIPS-20</td>
<td>TACAS-20, AAAI-21</td>
</tr>
<tr>
<td>CNF-XOR AAAI-19, CAV-20</td>
<td>Pseudo-Boolean CP-21</td>
<td>MaxSAT-XOR KR-21</td>
</tr>
<tr>
<td>Pseudo-Boolean CP-21</td>
<td></td>
<td>Hardware Accelerator</td>
</tr>
<tr>
<td>Hardware Accelerator</td>
<td></td>
<td>SAT-21</td>
</tr>
</tbody>
</table>

Slide 23/26
Reliability of Critical Infrastructure Networks

Figure: Plantersville, SC

Timeout = 1000 seconds
Timeout = 1000 seconds
Timeout = 1000 seconds

Impact: The first theoretically sound estimates of resilience in power transmission networks of ten medium sized cities in US
Applications

approxmc
Approximate Model Counter
- C++
- 51
- 18

SharpTNI: Counting and Sampling Parsimonious Transmission Networks under a Weak Bottleneck
Palash Sarkar† and Mohammed El-Khatir†

Static Evaluation of Noninterference using Approximate Model Counting
Ziqiao Zhou Zhiyun Qian Michael K. Reiter Yingjian Zhang

Check before You Change: Preventing Correlated Failures in Service Updates
Ennan Zhai†, Ang Chen+, Ruzica Piskac*, Mahesh Balakrishnan†* Bingchuan Tian†, Bo Song*, Haoliang Zhang*

Automating the Development of Chosen Ciphertext Attacks
Gabrielle Beck, Maximilian Zinkus, and Matthew Green, Johns Hopkins University

A Study of the Learnability of Relational Properties
Model Counting Meets Machine Learning (MCML)
Muhammad Uzaman University of Texas at Austin, USA muhammaduzaman@utexas.edu
Wenzi Wang University of Texas at Austin, USA wenziw@utexas.edu
Kaiyuan Wang† Haris Vikalo Marlos Vasic University of Texas at Austin, USA vasic@utexas.edu

Quantifying Software Reliability via Model-Counting
Samuel Tauris@ and Alexander Weigel@

IN SEARCH FOR A SAT-FRIENDLY BINARIZED NEURAL NETWORK ARCHITECTURE
Nina Narodytska Hongjie Zhang*
Where do we go from here?
Where do we go from here?

B. Cook, 2022: Virtuous cycle: ...application areas drives more investment in foundational tools, while improvements in the foundational tools drive further applications. Around and around.
Where do we go from here?

B. Cook, 2022: Virtuous cycle: ...application areas drives more investment in foundational tools, while improvements in the foundational tools drive further applications. Around and around.

Today’s Counters $\approx$ SAT Solvers in early 2000s
Where do we go from here?

B. Cook, 2022: **Virtuous cycle**: ...application areas drives more investment in foundational tools, while improvements in the foundational tools drive further applications. **Around and around**.

Today’s Counters ≈ SAT Solvers in early 2000s

**The storm is coming**: Statistical systems are being integrated into our lives, the Pentium FDIV and Ariane 5 Rocket moments are inevitable if we do not act

**Mission 2028**: 100× Speedup for Counting to enable Quantitative Reasoning at Scale
Where do we go from here?

B. Cook, 2022: Virtuous cycle: ...application areas drives more investment in foundational tools, while improvements in the foundational tools drive further applications. Around and around.

Today’s Counters ≈ SAT Solvers in early 2000s

The storm is coming: Statistical systems are being integrated into our lives, the Pentium FDIV and Ariane 5 Rocket moments are inevitable if we do not act

Mission 2028: 100× Speedup for Counting to enable Quantitative Reasoning at Scale

Challenge Problems

Civil Engineering Rigorous resilience estimation for power grid of Los Angeles

Neural Network Verification Neural networks with 1M neurons

Software Engineering Information Flow analysis of programs with 10K lines of code
Where do we go from here?

B. Cook, 2022: **Virtuous cycle**: ...application areas drives more investment in foundational tools, while improvements in the foundational tools drive further applications. **Around and around**.

Today’s Counters $\approx$ SAT Solvers in early 2000s

**The storm is coming**: Statistical systems are being integrated into our lives, the Pentium FDIV and Ariane 5 Rocket moments are inevitable if we do not act

**Mission 2028**: $100\times$ Speedup for Counting to enable Quantitative Reasoning at Scale

**Challenge Problems**

- **Civil Engineering**  Rigorous resilience estimation for power grid of Los Angeles
- **Neural Network Verification**  Neural networks with 1M neurons
- **Software Engineering**  Information Flow analysis of programs with 10K lines of code

**The road to promised land**: Theory + Algorithms + Software Development
Where do we go from here?

B. Cook, 2022: **Virtuous cycle**: ...application areas drives more investment in foundational tools, while improvements in the foundational tools drive further applications. **Around and around**.

Today’s Counters $\approx$ SAT Solvers in early 2000s

**The storm is coming**: Statistical systems are being integrated into our lives, the Pentium FDIV and Ariane 5 Rocket moments are inevitable if we do not act

**Mission 2028**: $100 \times$ Speedup for Counting to enable Quantitative Reasoning at Scale

**Challenge Problems**

**Civil Engineering**  Rigorous resilience estimation for power grid of Los Angeles

**Neural Network Verification**  Neural networks with 1M neurons

**Software Engineering**  Information Flow analysis of programs with 10K lines of code

**The road to promised land**: Theory + Algorithms + Software Development

**Where to start?:** Here! At IIT Bombay. **IIT Bombay Formal Methods group is hiring!**

These slides are available at tinyurl.com/meel-talk