The Eager Approach to SMT

Sanjit A. Seshia
UC Berkeley

Slides based on ICCAD ’09 Tutorial

Eager Approach to SMT

Input Formula

Satisfiability-preserving Boolean Encoder

Boolean Formula

SAT Solver

Key Ideas:
- Small-domain encoding
  - Constrain model search
- Rewrite rules
- Abstraction-based methods (eager + lazy)

Example Solvers:
UCLID, STP, Spear,
Boolector, Beaver, …
Theories

- Eager Encoding Methods have been demonstrated for the following Theories:
  - Equality & Uninterpreted Functions
  - Integer Linear Arithmetic
  - Restricted Lambda expressions
    - Arrays, memories, etc.
  - Finite-precision Bit-Vector Arithmetic
  - Strings

UCLID Operation

- Operation
  - Series of transformations leading to Boolean formula
  - Each step is validity (satisfiability) preserving
  - Each step performs optimizations

http://uclid.eecs.berkeley.edu
Rewrites: Eliminating Function Applications

- Two applications of an uninterpreted function \( f \) in a formula
- \( f(x_1) \) and \( f(x_2) \)

<table>
<thead>
<tr>
<th>Ackermann's Encoding</th>
<th>Bryant, German, Velev's Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f(x_1) )</td>
<td>( f(x_1) )</td>
</tr>
<tr>
<td>( f(x_2) )</td>
<td>( f(x_2) )</td>
</tr>
<tr>
<td>( x_1 = x_2 \Rightarrow vf_1 = vf_2 )</td>
<td>( ITE(x_1 = x_2, vf_1, vf_2) )</td>
</tr>
</tbody>
</table>

The Small Model Property

- A Theory is said to have the small-model property if, given a formula \( \phi \), \( \phi \) is satisfiable over the original domain \( D \) iff it is satisfiable over a domain \( S \), where \( S \) is a function of \( \phi \), and \( |S| < |D| \).
- Example: Integer Linear Arithmetic (QF_LIA)
**Small-Domain Encoding**

- Consider an SMT formula $\phi(x_1, x_2, \ldots, x_n)$ where $x_i \in D_i$

- Small-domain encoding/Finite instantiation: Derive finite set $S_i \subset D_i$ s.t. $|S_i| \ll |D_i|$
  - In some cases, $S_i$ is finite where $D_i$ is infinite

- Encode each $x_i$ to take values only in $S_i$
  - Could be done by encoding to SAT

---

**Solving QF_LIA is NP-complete**

- In NP:
  - If a satisfying solution exists, then one exists within a bound $d$
    - $\log d$ is polynomial in input size
  - Expression for $d$ [Papadimitriou, '82]
    $$(n+m) \cdot (b_{\text{max}} + 1) \cdot (m \cdot a_{\text{max}})^{2m+3}$$

- Input size:
  - $m$ – # constraints
  - $n$ – # variables
  - $b_{\text{max}}$ – largest constant (absolute value)
  - $a_{\text{max}}$ – largest coefficient (absolute value)
**Small-domain encoding / Finite Instantiation: Naïve approach**

- **Steps**
  - Calculate the solution bound $d$
  - Encode each integer variable with $\lceil \log d \rceil$ bits & translate to Boolean formula
  - Run SAT solver

- **Problem:** For QF_LIA, $d$ is $\Omega(m^m)$
  - $\Omega(m \log m)$ bits per variable

- **Solution:** Exploit special-cases and domain-specific structure

---

**Special Case 1: Equality Logic**

- Linear constraints are equalities $x_i = x_j$
- **Result:** $d = n$

$$x_1 \neq x_2 \land x_2 \neq x_3 \land x_1 \neq x_3$$

3-valued domain is needed: $\{1, 2, 3\}$

$$x_1 = x_2 \land x_2 \neq x_3 \land x_1 \neq x_3$$

Can find solution with domain $\{1, 2\}$

[PNueli et al., Information and Computation, 2002]
Special Case 2: Difference Logic

- Boolean combination of difference-bound constraints
  - \( x_i \geq x_j + b, \pm x_i \geq b \)
- Result: \( d = n \cdot (b_{\text{max}} + 1) \)
  [Bryant, Lahiri, Seshia, CAV’02]
- Proof sketch: satisfying solution corresponds to shortest path in constraint graph
  - Longest such path has length \( \leq n \cdot (b_{\text{max}} + 1) \)
- Tighter formula-specific bounds possible

Special Case 3: Generalized 2SAT

- Generalized 2SAT constraints
  - \( x_i + x_j \geq b, -x_i - x_j \geq b, x_i - x_j \geq b, x_i \geq b \)
- \( d = 2 \cdot n \cdot (b_{\text{max}} + 1) \)
  [Seshia, Subramani, Bryant,’04]
**Full Integer Linear Arithmetic**

- Can we avoid the $m^m$ blow-up?
- In fact, yes. The idea is to derive a new parameterized solution bound $d$
  - Formalize parameters that the bound really depends on
  - Parameters characterize sparse structure
    - Occurs especially in software verification; also in many high-level hardware models
  - [Seshia & Bryant, LICS’04, LMCS’05]

**Structure of Linear Constraints in Software Verification**

- Characteristics of studied benchmarks
  - Mostly difference constraints
    - Only 3% of constraints were NOT difference constraints
  - Non-difference constraints are sparse
    - At most 6 variables per constraint (total number of variables in 1000s)
- Some similar observations: Pratt’77, ESC/Java-Simplify-TR’03
Parameterized Solution Bound

- New parameters:
  - $k$ non-difference constraints,
  - $w$ variables per constraint (width)

- Our solution bound:
  \[ n \cdot (b_{\text{max}} + 1) \cdot (w \cdot a_{\text{max}})^k \]

Previous:
\[ (n+m) \cdot (b_{\text{max}} + 1) \cdot (m \cdot a_{\text{max}})^{2m+3} \]

- Direct dependence on $m$ eliminated (and $k \ll m$)

Example

\[ x_1 - x_2 \geq 1 \]
\[ x_1 + 2x_2 + x_3 > -3 \]
\[ x_2 - x_4 \geq 0 \]

<table>
<thead>
<tr>
<th>$m$</th>
<th>#constraints</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k$</td>
<td>#non-difference</td>
<td>1</td>
</tr>
<tr>
<td>$n$</td>
<td>#variables</td>
<td>4</td>
</tr>
<tr>
<td>$w$</td>
<td>width</td>
<td>3</td>
</tr>
<tr>
<td>$b_{\text{max}}$</td>
<td>max</td>
<td>3</td>
</tr>
<tr>
<td>$a_{\text{max}}$</td>
<td>max</td>
<td>2</td>
</tr>
</tbody>
</table>

$d = 96$

Previous $d$ = 282,175,488
### Summary of $d$ Values

<table>
<thead>
<tr>
<th>Logic</th>
<th>Solution Bound $d$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equality logic</td>
<td>$n$</td>
</tr>
<tr>
<td>Difference logic</td>
<td>$n \cdot (b_{\text{max}} + 1)$</td>
</tr>
<tr>
<td>Generalized 2SAT logic</td>
<td>$2 \cdot n \cdot (b_{\text{max}} + 1)$</td>
</tr>
<tr>
<td>Full Integer Linear Arithmetic</td>
<td>$n \cdot (b_{\text{max}} + 1) \cdot (a_{\text{max}}^k \cdot w^k)$</td>
</tr>
</tbody>
</table>

### Proof of Our Bound: Steps

1. Previous result for integer linear programming (ILP)
   - by Borosh-Treybig-Flahive [76, 86]

2. Express above result in $k$ and $w$, in addition to other parameters

3. Derive QFP bound from ILP bound
### Integer Linear Programming (ILP)

**Notation**

\[ Ax = b, \quad x \geq 0 \]

\[
\begin{bmatrix}
  a_{11} & a_{12} & \cdots & a_{1n} \\
  a_{21} & a_{22} & \cdots & a_{2n} \\
  \vdots & \vdots & \ddots & \vdots \\
  a_{m1} & a_{m2} & \cdots & a_{mn}
\end{bmatrix}
\cdot
\begin{bmatrix}
  x_1 \\
  x_2 \\
  \vdots \\
  x_n
\end{bmatrix}
= 
\begin{bmatrix}
  b_1 \\
  b_2 \\
  \vdots \\
  b_m
\end{bmatrix}
\]

\[ a_{\text{max}} = \max_{ij} |a_{ij}| \quad b_{\text{max}} = \max_i |b_i| \]

### Borosh-Treybig-Flahive Result [1986]

- Solution bound \( d \) is
  \[ (n+2) \cdot \Delta \]
  where \( \Delta = \text{largest sub-determinant of } [A \mid b] \text{ (abs. value)} \)

- Problem: Exponentially many sub-determinants!
Matrix Structure

\[ w \text{ non-zeroes per row} \]

\[ k \]

\[ m \]

\[ n \]

Non-difference constraints

\[ k = 0 \]: Only Difference Constraints

\[ x_i - x_j \geq b, \pm x_i \geq b \]

\[
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & b_1 \\
0 & 1 & -1 & 0 & 0 & b_2 \\
0 & 0 & -1 & 0 & 1 & b_3 \\
-1 & 0 & 0 & 1 & 0 & b_4 \\
0 & 1 & 0 & 0 & -1 & b_5 \\
\end{bmatrix}
\]

Totally Unimodular: All subdeterminants are in \{0, -1, +1\}

\[ \Delta \leq \sum_i |b_i| \leq \min(n+1, m) \cdot b_{\text{max}} \]
### Arbitrary $k$

Each term $\leq a_{\text{max}}^k$  

$\#\text{Terms} \leq w^k$

$$\Delta \leq \sum |b_i| (a_{\text{max}}^k \cdot w^k)$$

$$\leq \min(n+1, m) \cdot b_{\text{max}} \cdot (a_{\text{max}}^k \cdot w^k)$$

### Bound for ILP

- $\Delta \leq \min(n+1, m) \cdot b_{\text{max}} \cdot (a_{\text{max}}^k \cdot w^k)$

- $d = (n+2) \cdot \Delta$  
  [Borosh-Treybig-Flahive]
  $$= (n+2) \cdot \min(n+1, m) \cdot b_{\text{max}} \cdot (a_{\text{max}}^k \cdot w^k)$$
  $$\leq (n+2) \cdot n \cdot b_{\text{max}} \cdot (a_{\text{max}}^k \cdot w^k)$$
  (assuming $m \leq n$)
QFP Bound from ILP Bound

- Consider DNF of arbitrary QFP formula $\phi$
  $$\phi = \phi_1 \lor \phi_2 \lor \ldots \lor \phi_N$$

- Satisfying assignment to $\phi$ must satisfy some $\phi_i$

- Each $\phi_i$ is an ILP
  - Parameters of $\phi_i$ are bounded by those of $\phi$

- Therefore: $d = (n+2) \cdot n \cdot b_{\text{max}} \cdot (a_{\text{max}}^k \cdot w^k)$

Summary of $d$ Values

<table>
<thead>
<tr>
<th>Logic</th>
<th>$d$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equality logic</td>
<td>$n$</td>
</tr>
<tr>
<td>Separation logic</td>
<td>$n \cdot (b_{\text{max}} + 1)$</td>
</tr>
<tr>
<td>Generalized 2SAT logic</td>
<td>$2 \cdot n \cdot (b_{\text{max}} + 1)$</td>
</tr>
<tr>
<td>Quantifier-Free Presburger logic</td>
<td>$(n+2) \cdot n \cdot (b_{\text{max}} + 1) \cdot (a_{\text{max}}^k \cdot w^k)$</td>
</tr>
</tbody>
</table>

Note: A paper by Sergei Veselov proved that the bound of Borosh-Treybig-Flahive has been improved from $(n+2) \cdot \Delta$ to simply $\Delta$. This eliminates the $(n+2)$ term from the last entry in the table above.
**Abstraction-Based Methods**

- For some logics, one cannot easily compute a closed-form expression for the small domain
- **Example:** Bit-Vector Arithmetic
- In such cases, an abstraction-refinement approach can be used to compute formula-specific small domains

---

**Bit-Vector Arithmetic: Some History**

- **B.C. (Before Chaff)**
  - String operations (concatenate, field extraction)
  - Linear arithmetic with bounds checking
  - Modular arithmetic
- **SAT-Based “Bit Blasting”**
  - Generate Boolean circuit based on bit-level behavior of operations
    - Handles arbitrary operations
  - Check with best available SAT solver
  - Effective in many applications
    - CBMC [Clarke, Kroening, Lerda, TACAS ’04]
    - Microsoft Cogent + SLAM [Cook, Kroening, Sharygina, CAV ’05]
Research Challenge

• *Is there a better way than bit blasting?*

• **Requirements**
  – Provide same functionality as with bit blasting
    • Must support all bit-vector operators
  – Exploit word-level structure
  – Improve on performance of bit blasting

• **Current Approaches based on two core ideas:**
  1. **Simplification:** Simplify input formula using word-level rewrite rules and solvers
  2. **Abstraction:** Can use automatic abstraction-refinement to solve simplified formula

Bit-Vector SMT Solvers, circa Spr.’2009

Current Techniques with Sample Tools

– *Proof-based abstraction-refinement* – **UCLID** [Bryant et al., TACAS ’07]
– *Solver for linear modular arithmetic* to simplify the formula – **STP** [Ganesh & Dill, CAV’07]
– *Automatic parameter tuning for SAT* – **Spear** [Hutter et al., FMCAD ’07]
– *Rewrites, underapproximation, efficient SAT engine* – **Boolector** [Brummayer & Biere, TACAS’09]
– *Equality/constant propagation, logic optimization, special rules for non-linear ops* - **Beaver** [Jha et al., CAV’09]
– *DPLL(T) framework: Layered approach, rewriting* – **CVC3** [Barrett et al.], **MathSAT** [Bruttomesso et al.], **Yices** [Dutertre et al.], **Z3** [de Moura et al.]
**Abstraction-Refinement**

- Deciding Bit-Vector Arithmetic with Abstraction [Bryant et al., TACAS ’07, STTT ’09]
  - Use bit blasting as core technique
  - Apply to simplified versions of formula: under and over approximations
  - Generate successive approximations until a solution is found or formula shown unsatisfiable
  - Inspired by McMillan & Amla’s proof-based abstraction for finite-state model checking
- Small Motivating Example:
  \[(x + y \neq y + x) \land (x \cdot y \neq y \cdot x)\]
  - Sufficient to prove the left-hand conjunct unsat

**Approximations to Formula**

- Overapproximation \(\varphi^+\)
  - \(\varphi \Rightarrow \varphi^+\)
  - More solutions: If unsatisfiable, then so is \(\varphi\)

- Original Formula \(\varphi\)

- Underapproximation \(\varphi^-\)
  - \(\varphi^- \Rightarrow \varphi\)
  - Fewer solutions: Satisfying solution also satisfies \(\varphi\)

- Example Approximation Techniques
  - Underapproximating
    - Restrict word-level variables to smaller ranges of values
  - Overapproximating
    - Replace subformula with Boolean variable
Starting Iterations

- Initial Underapproximation
  - (Greatly) restrict ranges of word-level variables
  - Intuition: Satisfiable formula often has small-domain solution

First Half of Iteration

- SAT Result for $\varphi_1^-$
  - Satisfiable
    - Then have found solution for $\varphi$
  - Unsatisfiable
    - Use UNSAT proof to generate overapproximation $\varphi_1^+$
Second Half of Iteration

- SAT Result for $\varphi_1^+$
  - Unsatisfiable: then have shown $\varphi$ unsatisfiable
  - Satisfiable: solution indicates variable ranges that must be expanded
- Generate refined underapproximation

Example

$\varphi_1^+ := (x = y+2)$

$\varphi := (x = y+2) \land (x^2 > y^2)$

$\varphi_2^- := (x_{[2]} = y_{[2]} + 2) \land (x_{[2]}^2 > y_{[2]}^2)$

$\varphi_1^- := (x_{[1]} = y_{[1]} + 2) \land (x_{[1]}^2 > y_{[1]}^2)$

SAT: $x = 2, y = 0$

SAT: done.

UNSAT: Look at proof
**Iterative Behavior**

- **Underapproximations**
  - Successively more precise abstractions of $\varphi$
  - Allow wider variable ranges
- **Overapproximations**
  - No predictable relation
  - UNSAT proof not unique

**Overall Effect**

- **Soundness**
  - Only terminate with solution on underapproximation
  - Only terminate as UNSAT on overapproximation
- **Completeness**
  - Successive underapproximations approach $\varphi$
  - Finite variable ranges guarantee termination
  - In worst case, get $\varphi_k^- = \varphi$
Summary of Key Ideas

Summary of Ideas: Lazy Methods

- Philosophy: Extend DPLL framework from SAT to SMT
- Literals assigned by SAT are sent to Theory Solver
- Theory Solver determines if literals are satisfiable in the theory
- Key optimizations: small explanations, early conflict detection, theory propagation
Summary of Ideas: Eager Methods

- Philosophy: Constrain solution space with theory-specific solution bounds
- Small-domain encoding
  - Compute bounds that work for any formula in the logic
- Abstraction-refinement of domains
  - Compute formula-specific small domains
- Rewrite rules: high level and bit level
  - Simplify formula before and after bit-blasting

Challenges and Opportunities

- Solvers for new theories
  - Strings
  - Non-linear arithmetic
  - Can we exploit domain-specific structure?
- Parallel SMT
- Better support for quantifiers
- Better proof/interpolant generation