Abstraction and Multi-Encodings in SAT

Carsten Sinz

Department for Theoretical Computer Science (ITI)
Karlsruhe Institute of Technology (KIT)

January 24, 2014
Abstraction: Idea & Motivation

• **Question:** Does $S \Rightarrow P$ hold?
  • $S$: System
  • $P$: Property

• **Applications:**
  • Model Checking, Software Verification, ...

• **Observations:**
  • Large state space (large formulas)
  • Only small part of $S$ might be needed to show $P$

• **Abstraction:**
  • Replace $S$ (or $P$) by approximate formalization $S'$ (or $P'$)
    $\Rightarrow$ Hope that $S' \Rightarrow P$ is easier to prove than $S \Rightarrow P$

• **In what follows:** "F SAT/UNSAT" instead of "$S \Rightarrow P$ valid"
Abstraction

• Two variants of abstraction:
  • Under-approximation: abstraction has fewer solutions
  • Over-approximation: abstraction has more solutions

• Definitions:
  • $F'$ is an under-approximation of $F$ if: $\alpha \models F'$ implies $\alpha \models F$
  • $F'$ is an over-approximation of $F$ if: $\alpha \not\models F$ implies $\alpha \not\models F'$

• Simple properties:
  • For all $F$: $F' = \text{false}$ is an under-approximation of $F$
  • For all $F$: $F' = \text{true}$ is an over-approximation of $F$

• Abstraction is an extremely successful technique in model checking and (bit-blasting) SMT solvers

• Refinement: "better" approximation

• Refinement loop: gradually compute improved approximations
Under-Approximation

\[ F \quad \text{"Problem"} \rightarrow \text{Initial Abstraction} \rightarrow \text{Solve} \rightarrow \text{Refine Abstraction} \]

- SAT? yes \rightarrow SAT
- SAT? no \rightarrow Refine Abstraction
- Proof also for Orig. Prob.? yes \rightarrow UNSAT
- Proof also for Orig. Prob.? no \rightarrow SAT?
Over-Approximation

- F "Problem"
- Initial Abstraction
- Solve
- Refine Abstraction
- "spurious counterexample"
- UNSAT
- SAT?
- SAT

CEGAR
Counterexample-guided
Abstraction Refinement
CEGAR: What do we need?

- **Initial abstraction**
  - Should be close to problem, but easy to solve

- **Checking abstract counterexample**

- **Abstraction refinement**
  - Better approximation
  - Based on counterexample

- **Example: Check, whether F (in CNF) is satisfiable**
  - **Initial abstraction**: 2-clauses of F
  - **Checking abstract counterexample**: all clauses of F satisfied?
  - **Refinement**: add all clauses not satisfied by counterexample
CEGAR: Example

\[ F = \{ \{ a, b, \neg c \}, \{ a, c \}, \{ \neg a, b \}, \{ \neg b, \neg d \}, \{ \neg a, b, \neg c, \neg d \}, \{ \neg b, d \} \} \]

1. Initial approximation: (2-clauses of \( F \))
   \[ F_0^{\text{over}} = \{ \{ a, c \}, \{ \neg a, b \}, \{ \neg b, \neg d \}, \{ \neg b, d \} \} \]

2. Solve: SAT, \( \alpha = \{ \neg a, \neg b, c, \neg d \} \)

3. Check \( \alpha \) for \( F \): \( \alpha \not\models F \)

4. Refine:
   \[ F_1^{\text{over}} = \{ \{ a, c \}, \{ \neg a, b \}, \{ \neg b, \neg d \}, \{ \neg b, d \}, \{ a, b, \neg c \} \} \]

5. Solve: UNSAT

\[ \rightarrow \] Clause \{ \neg a, b, \neg c, \neg d \} not needed

\[ \rightarrow \] CEGAR improves "locality"
Abstraction in SAT

• Abstraction not widely employed in SAT solving
  • Exceptions: bounded model checking, SMT solvers

• Why?

<table>
<thead>
<tr>
<th></th>
<th>Total</th>
<th>MiniSAT</th>
<th>MiniSAT + Abstraction</th>
<th>MiniSAT + 10 Abstr. Rounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAT-Race 2010</td>
<td>100</td>
<td>68</td>
<td>48</td>
<td>67</td>
</tr>
</tbody>
</table>

• Reasons:
  • Abstraction-refinement approach in experiment too simple
  • Not enough high-level information for effective abstraction-refinement
Where Abstraction in SAT Works

- SMT solver, theory of arrays
- McCarthy (read-over-write) axiom:

\[
\text{read}(\text{write}(a, i, x), j) = \begin{cases} 
  x & \text{falls } i = j \\
  \text{read}(a, i) & \text{sonst}
\end{cases}
\]

- Approximation: do not encode functional consistency for read

**Example:** array logic formula \( F \):

\[
\text{read}(a, i) = x \land \text{read}(a, j) = y \land i = j \land x \neq y
\]

- SAT encoding: (with 1-bit bit vectors)

\[
\{ \{ \neg r_{a,i}, x \}, \{ r_{a,i}, \neg x \}, \{ \neg r_{a,j}, y \}, \{ r_{a,j}, \neg y \}, \{ i, \neg j \}, \{ i, j \}, \{ x, y \}, \{ \neg x, \neg y \}, \\
\{ \neg i, \neg j, \neg r_{a,i}, r_{a,j} \}, \{ \neg i, \neg j, r_{a,i}, \neg r_{a,j} \}, \{ i, j, \neg r_{a,i}, r_{a,j} \}, \{ i, j, r_{a,i}, \neg r_{a,j} \} \}
\]

- Initial abstraction:

\[
F_{0}^{\text{over}} = \{ \{ \neg r_{a,i}, x \}, \{ r_{a,i}, \neg x \}, \{ \neg r_{a,i}, y \}, \{ r_{a,i}, \neg y \}, \{ i, \neg j \}, \{ i, j \}, \{ x, y \}, \{ \neg x, \neg y \} \}
\]
Where Abstraction in SAT Works (II)

- Modulo-Operation in bit-blasting SMT solver
  \[ z = x \mod y \] (bvurem, truncated division)
  - Encoded precisely using a circuit (e.g. for 32-bit signed integers)
- **Possible abstractions:** replace \( z = x \mod y \) by
  - \( 0 \leq z < y \)
  - \( z = x \gg k \), if \( y = 2^k \), where \( k \) is constant
  - \( x < y \Rightarrow z = x \)
  - \( y \leq x < 2y \Rightarrow z = x - y \)
Abstraction in SAT

- Abstraction seems to work, if high-level information can be used, e.g.
  - Groups of clauses that encode the same high-level constraint
  - Semantics of group of clauses that can be abstracted
- **Extend SAT solvers?**
  - General abstraction-based SAT engine
  - Use high-level information about problem for abstractions
- **Possible ways to realize:**
  - Mark groups of clauses by "abstraction levels"
  - Provide different encodings ("**Multi-encoding**")
    - E.g., different ways to encode a cardinality constraint
  - High-level description language
Current SAT Solvers

[Adapted from Biere: Understanding Modern SAT Solvers]
Extended SAT Solvers

High-level Encode

Low-level Encode

Simplify

Search
Summary

- Abstraction techniques successful in certain SAT applications
- **Research direction:** make abstractions broadly applicable by
  - Passing high-level problem information to SAT solver
  - Using multiple encodings simultaneously
- **Does it also help in proof complexity research? Maybe**
  - Automatizability of proof procedures
  - Analyze SAT in conjunction with problem encoding
  - Shorter proofs by using multiple encodings simultaneously?