Engineering Satisfiability Modulo Theories Solvers for Intractable Problems

Nikolaj Bjørner
Microsoft Research
Tractability Workshop – MSR Cambridge July 5,6 2010
This talk

Z3 – An Efficient SMT solver: Overview and Applications.

A “hands on” example of Engineering SMT solvers: Efficient Theory Resolution using DPLL(T).
Some Microsoft Engines using Z3

- **SDV:** The Static Driver Verifier
- **PREfix:** The Static Analysis Engine for C/C++.
- **Pex:** Program EXploration for .NET.
- **SAGE:** Scalable Automated Guided Execution
- **Spec#:** C# + contracts
- **VCC:** Verifying C Compiler for the Viridian Hyper-Visor
- **HAVOC:** Heap Aware Verification of C-code.
- **SpecExplorer:** Model-based testing of protocol specs.
- **Yogi:** Dynamic symbolic execution + abstraction.
- **FORMULA:** Model-based Design
- **F7:** Refinement types for security protocols
- **M3:** Model Program Modeling
- **VS3:** Abstract interpretation and Synthesis
- **VERVE:** Verified operating system
- **FINE:** Proof-carrying certified code
SAGE by the numbers

Slide shamelessly stolen and adapted from [Patrice Godefroid, ISSTA 2010]

100+ CPU-years - largest dedicated fuzz lab in the world

100s apps - fuzzed using SAGE

100s previously unknown bugs found

1,000,000,000+ computers updated with bug fixes

Millions of $ saved for Users and Microsoft

10s of related tools (incl. Pex), 100s DART citations

100,000,000+ constraints - largest usage for any SMT solver
```c
int binary_search(int arr[], int low, int high, int key)
{
    while (low <= high)
    {
        // Find middle value
        int mid = (low + high) / 2;
        int val = arr[mid];
        if (val == key) return mid;
        if (val < key) low = mid+1;
        else high = mid-1;
    }
    return -1;
}
```

- **Function**: `itoa` (integer to ascii)
- **Package**: `java.util.Arrays`
- **Book**: Kernighan and Ritchie

**Note**: The value `-INT_MIN` is calculated as `3(INT_MAX+1)/4 + (INT_MAX+1)/4 = INT_MIN`. This is used to handle negative numbers when converting integers to strings.
Example: an overflowed allocation size

ULONG AllocationSize;
while (CurrentBuffer != NULL) {
    if (NumberOfBuffers > MAX_ULONG / sizeof(MYBUFFER)) {
        return NULL;
    }
    NumberOfBuffers++;
    CurrentBuffer = CurrentBuffer->NextBuffer;
}
AllocationSize = sizeof(MYBUFFER)*NumberOfBuffers;
UserBuffersHead = malloc(AllocationSize);

Bug is simple and local within a large program

... Overflow((nb+1)*sizeof(MYBUFFER))
CurrentBuffer == NULL
nb <= MAX_ULONG/sizeof(MYBUFFER)
Building Verve

9 person-months

Source file
Verification tool
Compilation tool

Kernel.cs
C# compiler

Translator/Assembler
Boogie/Z3

Kernel.obj (x86)
TAL checker

Nucleus.bpl (x86)
Linker/ISO generator

Verve.iso

Safe to the Last Instruction / Jean Yang & Chris Hawbliztl
PLDI 2010
What is Satisfiability Modulo Theories?

$x + 2 = y \Rightarrow f(\text{read}\{(\text{write}(a, x, 3), y - 2)\}) = f(y - x + 1)$

Array Theory

Arithmetic

Uninterpreted Functions

read(write(a, i, v), i) = v

i ≠ j ⇒ read(write(a, i, v), j) = read(a, j)
What is Z3?

Theory Solvers
- Bit-Vectors
- Lin-arithmetic
- Arrays
- Groebner basis
- Recursive Datatypes
- Comb. Array Logic
- Free (uninterpreted) functions

Simplify
SMT-LIB
Native

OCaml
.NET
C
F# quote

Model Generation:
- Finite Models

Quantifiers:
- E-matching
- Super-position

SAT core

Parallel Z3

Proof objects

Cores: Assumption tracking

By Leonardo de Moura & Nikolaj Bjørner [http://research.microsoft.com/projects/z3](http://research.microsoft.com/projects/z3)
Constraints from Software Applications are in spite of Constraint language highly intractable Algorithms high worst case complexity Tractable
Modification in invariant checking

Switch to Boogie2

Switch to Z3 v2

Z3 v2 update

Attempt to improve Boogie/Z3 interaction

Viel Spaß und liebe Grüße an Lieven,
Markus
Constraint languages highly intractable

Algorithms high worst case complexity
Tractability and Applications

Constraints from Software Applications are Tractable

\[ a \leq b \land \]
\[ b < c \land \]
\[ c \leq a \land \]
\[ x \leq y \land \]
\[ y < z \land \]
\[ z < u \land \]
\[ x \leq w \land \]
\[ x \leq v \land \]
\[ x \leq 1 \land \]
\[ x \leq 2 \land \]
\[ x \leq 3 \]

Unsat

\[ a \leq b \land \]
\[ b \leq c \land \]
\[ c \leq a \land \]
\[ x = w \land \]
\[ x = v \land \]
\[ x = 1 \land \]
\[ x \leq 2 \land \]
\[ x \leq 3 \land \]
\[ x \leq y \land \]
\[ y < z \land \]
\[ z < u \land \]

\[ a = b = c \]
\[ x, v, w = 1 \]
\[ x = 1 \leq 2,3 \]
\[ y,z,u \text{ “free”} \]

Proofs are small

Models are determined or free
What is then important for engineering solvers?

- Solve tractable parts
  - efficient theory solvers
- Strong Simplification
  - reduce the clutter
- Efficient Indexing
  - minimize & reuse work
- Avoid getting stuck
  - restarts, parallel search
What is then important for engineering solvers?

<table>
<thead>
<tr>
<th>What</th>
<th>Important Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solve tractable parts</td>
<td>- efficient theory solvers</td>
</tr>
<tr>
<td></td>
<td>[Efficient, Generalized Array Decision Procedures de Moura &amp; B]</td>
</tr>
<tr>
<td>Strong Simplification</td>
<td>- reduce the clutter</td>
</tr>
<tr>
<td></td>
<td>[Z3 An Efficient SMT Solver de Moura &amp; B]</td>
</tr>
<tr>
<td>Efficient Indexing</td>
<td>- minimize &amp; reuse work</td>
</tr>
<tr>
<td></td>
<td>[Efficient E-matching de Moura &amp; B]</td>
</tr>
<tr>
<td>Avoid getting stuck</td>
<td>- restarts, parallel search</td>
</tr>
<tr>
<td></td>
<td>[Parallel Portfolio, Wintersteiger, Hamadi &amp; de Moura]</td>
</tr>
</tbody>
</table>
Tractability and Applications

Constraints from Software Applications are Tractable

Problem solved, end of talk
Constraints from Software Applications are Tractable

sometimes quite intractable for existing techniques
Symptom of a problem

```java
public void Diamond(int a) {
    if (p1(a))
        a++;
    else
        a--;
}
...
if (p100(a))
    a++;  
else
    a--;
}

\[
\begin{align*}
& (p_1(a_0) \land a_1 \approx a_0 + 1) \\
& \lor (\neg p_1(a_0) \land a_1 \approx a_0 - 1) \\
& \land (p_2(a_1) \land a_2 \approx a_1 + 1) \\
& \lor (\neg p_2(a_1) \land a_2 \approx a_1 - 1) \\
& \land \ldots \\
& \land (p_{100}(a_{99}) \land a_{100} \approx a_{99} + 1) \\
& \lor (\neg p_{101}(a_{99}) \land a_{100} \approx a_{99} - 1) \\
& \rightarrow a_0 - 100 \leq a_{100} \leq a_0 + 100
\end{align*}
\]

assert(\text{old}(a) - 100 \leq a \leq \text{old}(a) + 100);
```

Poses a challenge to Z3
Another challenge

Bit-vector multiplication using SAT

\[ \text{out}_0 = a_0 \times b_0 \]
\[ \text{out}_1 = a_0 \times b_1 \]
\[ \text{out}_2 = a_0 \times b_2 \]
\[ \text{out}_3 = a_0 \times b_3 \]

\( O(n^2) \) clauses

SAT solving time increases exponentially. Similar for BDDs. [Bryant, MC25, 08]

Brute-force enumeration + evaluation faster for 20 bits. [Matthews, BPR 08]
A Framework and its limitations

DPLL(T) is Z3’s main core search framework

**Efficient SAT technologies**
- DPLL + CDCL + Restart = Space Efficient Resolution

**Efficient integration of incremental theory solvers**
- Theory lemmas (T-Conflicts)
- Theory propagation (T-Propagation)

**But we claim**
- Contemporary DPLL(T) < Resolution
But ... $DPLL(T) < \text{Resolution}$

Possible remedies:

- Forget $DPLL(T)$. Use other core engine.
- Adapt $DPLL(T)$. Elaboration here. We call it:

  **Conflict Directed Theory Resolution**
Review: SAT made “tractable”

Propagate $\neg p$

Guess $q$

Resolve
Learn $\neg q$

Conflict

$\neg p \lor \neg q, \ p \lor \neg q, \ \neg p \lor q, \ p \lor q$

Propagate $\neg q$

Backjump

$\bot$
Review: SAT made “tractable”

- Builds resolution proof
  - General Resolution \(\equiv\) DPLL + CDCL + Restart
    (CDCL: Conflict Directed Clause Learning)

- Space Efficient
  - DPLL does not create intermediary clauses

- Efficient indexing and heuristics
  - 2-watch literals, Restarts, phase selection, clause minimization
# Review: Modern DPLL in a nutshell

<table>
<thead>
<tr>
<th>Operation</th>
<th>Expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialize</td>
<td>$\epsilon</td>
<td>F$</td>
</tr>
<tr>
<td>Decide</td>
<td>$M</td>
<td>F \Rightarrow M, \ell</td>
</tr>
<tr>
<td>Propagate</td>
<td>$M</td>
<td>F, C \lor \ell \Rightarrow M, \ell^{C\lor \ell}</td>
</tr>
<tr>
<td>Conflict</td>
<td>$M</td>
<td>F, C \Rightarrow M</td>
</tr>
<tr>
<td>Resolve</td>
<td>$M</td>
<td>F</td>
</tr>
<tr>
<td>Learn</td>
<td>$M</td>
<td>F</td>
</tr>
<tr>
<td>Backjump</td>
<td>$M \neg \ell M'</td>
<td>F</td>
</tr>
<tr>
<td>Unsat</td>
<td>$M</td>
<td>F</td>
</tr>
<tr>
<td>Sat</td>
<td>$M</td>
<td>F \Rightarrow M$</td>
</tr>
<tr>
<td>Restart</td>
<td>$M</td>
<td>F \Rightarrow \epsilon</td>
</tr>
</tbody>
</table>

Adapted and modified from [Nieuwenhuis, Oliveras, Tinelli J.ACM 06]
DPLL(T) in a nutshell

T- Propagate
\[ M \mid F, C \lor \ell \Rightarrow M, \ell^{C \lor \ell} \mid F, C \lor \ell \]
\[ \text{C is false under } T + M \]

T- Conflict
\[ M \mid F \Rightarrow M \mid F \mid \neg M' \]
\[ M' \subseteq M \text{ and } M' \text{is false under } T \]

T- Propagate
\[ a > b, b > c \mid F, a \leq c \lor b \leq d \Rightarrow \]
\[ a > b, b > c, b \leq a^{C \lor b \leq d} \mid F, a \leq c \lor b \leq d \]

T- Conflict
\[ M \mid F \Rightarrow M \mid F, a \leq b \lor b \leq c \lor c < a \]
\[ \text{where } a > b, b > c, a \leq c \subseteq M \]

Introduces no new literals - terminates
DPLL(T) misses short proofs

The **Black Diamonds** of DPLL(T)

\[ \neg(a_1 \approx a_{50}) \land \bigwedge_{i=1}^{49} [(a_i \approx b_i \land b_i \approx a_{i+1}) \lor (a_i \approx c_i \land c_i \approx a_{i+1})] \]

Has no short DPLL(T) proof.

Has short DPLL(T) proof when using \( a_1 \approx a_2, a_2 \approx a_3, a_3 \approx a_4, \ldots, a_{49} \approx a_{50} \)

Example from [Rozanov, Strichman, SMT 07]
DPLL(T) misses short proofs

Idea: DPLL(⊔)

Try branch $a_1 \approx b_1 \land b_1 \approx a_2$
Implies $a_1 \approx b_1 \approx a_2$
Collect implied equalities

Try branch $\neg(a_1 \approx b_1 \land b_1 \approx a_2)$
Implies $a_1 \approx c_1 \approx a_2$
Collect implied equalities

Compute the join $⊔$ of the two equalities – common equalities are learned

Still potentially $O(n^2)$ rounds just at base level of search.
DPLL(∪ base) misses short proofs

Single case splits don’t suffice

\[ a_1 \not= a_{50} \land \bigwedge_{i=1}^{49} \left[ \begin{array}{c} (a_i \equiv b_i \land b_i \equiv a_{i+1}) \\ \lor (a_i \equiv c_i \land c_i \equiv a_{i+1}) \\ \lor (a_i \equiv d_i \land d_i \equiv a_{i+1}) \end{array} \right] \]

Requires 2 case splits to collect implied equalities
We now describe an approach we call:

**Conflict Directed Theory Resolution**

resolve literals from conflicts
→ simulates resolution proofs.

Engineering: **Throttle** resolution dynamically based on activity.
Eventually, many conflicts contain:

$$\neg(a_1 \simeq a_{50}) \land \bigwedge_{i=1}^{49} [(a_i \simeq b_i \land b_i \simeq a_{i+1}) \lor (a_i \simeq c_i \land c_i \simeq a_{i+1})]$$

Use E-resolution, add clause:

$$a_1 \simeq b_1 \land b_1 \simeq a_2 \lor \cdots$$

Then DPLL(T) learns by itself:

$$a_1 \simeq a_2$$
Eventually, many conflicts contain:

\[ \bigwedge_{i=1}^{N} (p_i \lor x_i \simeq v_0) \land (\neg p_i \lor x_i \simeq v_1) \land (p_i \lor y_i \simeq v_0) \land (\neg p_i \lor y_i \simeq v_1) \land \\
\neg (f(x_N, \ldots, f(x_2, x_1) \ldots) \simeq f(y_N, \ldots, f(y_2, y_1) \ldots)) \]

Eventually, many conflicts contain:

\[ x_i \simeq u_i \land y_i \simeq u_i \land u_i = v_0 \text{ or } u_i = v_1 \text{ for } i = 1 \ldots N \land \\
\neg (f(x_N, \ldots, f(x_2, x_1) \ldots) \simeq f(y_N, \ldots, f(y_2, y_1) \ldots)) \]

Add:

\[ \bigwedge_{i=1}^{N} x_i \simeq y_i \rightarrow f(x_N, \ldots, f(x_2, x_1) \ldots) \simeq f(y_N, \ldots, f(y_2, y_1) \ldots) \]
Deciding Th(Equality)

\[ a = f(f(a)), a = f(f(f(a))), a \neq f(a) \]

First Step: “Naming” subterms
Deciding Th(Equality)

\[ a = v_2, a = v_3, a \neq v_1, \]
\[ v_1 \equiv f(a), v_2 \equiv f(v_1), v_3 \equiv f(v_2) \]

... and merge equalities
\( a = v_2, a = v_3, a \neq v_1, \\
v_1 \equiv f(a), v_2 \equiv f(v_1), v_3 \equiv f(v_2) \)

Second step. Apply Congruence Rule:
\( x_1 = y_1, \ldots, x_n = y_n \) implies \( f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n) \)
Deciding Th(Equality)

\[ a = v_2, a = v_3, a \neq v_1, \]
\[ v_1 \equiv f(a), v_2 \equiv f(v_1), v_3 \equiv f(v_2) \]

Second step. Apply Congruence Rule:
\[ a \simeq v_2 \implies f(a) \simeq f(v_2): \quad v_1 \simeq v_3 \]
Dynamic Ackermann Reduction

If Congruence Rule repeatedly learns

\[ f(v, v') \sim f(w, w') \]

Then add clause for SAT core to use

\[ v \simeq w \land v' \simeq w' \rightarrow f(v, v') \simeq f(w, w') \]

Used in Yices and Z3 to find short congruence closure proofs
[Yices Tool 06, Dutertre, de Moura]
[Model-based Theory Combination 07, de Moura, B]
Dynamic Ackermann Reduction

If Congruence Rule repeatedly learns

\[ f(v, v') \sim f(w, w') \]

for literal \( f(v, v') \approx f(w, w') \)

Then add clause for SAT core to use

\[ v \approx w \land v' \approx w' \rightarrow f(v, v') \approx f(w, w') \]

Leo identified the following useful optimization filter heuristic used in Z3

“Peel the onion from outside”
CDTR for Th(Equalities)

**Dynamic Ackermann Reduction**

If *Congruence Rule* repeatedly learns

\[ f(v, v') \sim f(w, w') \]

Then add clause for SAT core to use

\[ v \equiv w \land v' \equiv w' \rightarrow f(v, v') \equiv f(w, w') \]

**Dynamic Ackermann Reduction with Transitivity**

If *Equality Transitivity* repeatedly learns

\[ u \sim w \quad \text{from} \quad u \sim v \text{ and } v \sim w \]

Then add clause for SAT core to use

\[ u \equiv v \land v \equiv w \rightarrow v \equiv w \]
Claim: Ground E-Resolution

\[ \equiv \]

DPLL(E) + Dynamic Ackermann Reduction with Transitivity

Alternative: Static Ackermann Reduction

[Singerman, Pnueli, Velev, Bryant, Strichman, Lahiri, Seisha, Bruttomesso, Cimatti, Franzen, Griggio, Santuari, Sebastiani]

P-simulates ground E-Resolution.

But it has high up-front space overhead.
CDTR for Linear Difference Arithmetic

\[
\begin{align*}
a &< x_1 \land a < x_2 \land (x_1 < b \lor x_2 < b) \land \\
b &< y_1 \land b < y_2 \land (y_1 < c \lor y_2 < c) \land \\
c &< z_1 \land c < z_2 \land (z_1 < a \lor z_2 < a)
\end{align*}
\]
CDTR: Linear Difference Arithmetic
CDTR: Linear Difference Arithmetic
CDTR: Linear Difference Arithmetic
Top Two Most Active vertices

Add clause
\[ a < x_1 < b \rightarrow a < b \]
Z3 supported theories all reduce to one of

<table>
<thead>
<tr>
<th>Arithmetic</th>
<th>Equality</th>
<th>Booleans</th>
</tr>
</thead>
</table>

**CDTR**

- **Th(Equalities):** Extended Dynamic Ackermann
- **Th(Differences):** Cutting loops
- **Th(LRA):** Fourier-Motzkin resolution
- **Th(LIA):** Perhaps: Integer FM [B. IJCAR 10]

**CDTR and theory combinations:**

- Theories communicate equalities between shared variables.
- Build clauses using these equalities.
Modern SMT solvers are tuned to but limitations of basic proof calculus shows up.

Presented a technique to close the gap

**Dynamic** - to make it practical.

Based on applying **Resolution** to conflicts.

Just one of many possible optimizations.

The quest for improving search continues
e.g. cutting plane proofs, arbitrary cuts (Frege)