Scaling SMT Solving for Applications

Nikolaj Bjørner
Microsoft Research
Deduction at Scale, Schloß Ringberg March 7
Some Microsoft Engines using Z3

Try them online: http://rise4fun.com
using System;
using Microsoft.Pex.Framework;
[PexClass]
public class TestClass {
    // Which values will trigger collisions in MyHashSet? Ask Pex to find out!
    [PexMethod] // this puzzle is a 'Parameterized Unit Test'
    public void TestAddContains(int x, int y) {
        var s = new MyHashSet();
        s.Add(x);
        s.Add(y);
        PexAssert.IsTrue(s.Contains(x));
        PexAssert.IsTrue(s.Contains(y));
    }
}

class MyHashSet {

    // Ask Pex!
    Done. 6 interesting inputs found. How does Pex work?

    | Value 1 | Value 2 | Error Message |
    |--------|--------|---------------|
    | 0      | 0      | ArgumentException '0' not allowed |
    | 1      | 0      | ArgumentException '0' not allowed |
    | -704287306 | 0 | ArgumentException Index was outside the bounds of the array |
    | 485    | 700    | ArgumentException |
    | 43     | 690    | ArgumentException |
Rex – Regular Expression Exploration

Can you discover the secret regex? Click 'ask Rex'! Read more or watch the video.

You Missed! Your regex gave different matches than the secret regex. Try modifying it and Ask Rex again!

<table>
<thead>
<tr>
<th>string</th>
<th>your regex</th>
<th>secret regex</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;r&quot;</td>
<td>match</td>
<td>match</td>
<td></td>
</tr>
<tr>
<td>&quot;H691&quot;</td>
<td>match</td>
<td>match</td>
<td></td>
</tr>
<tr>
<td>&quot;F_5\n&quot;</td>
<td>match</td>
<td>no match</td>
<td></td>
</tr>
<tr>
<td>&quot;Q\n\n&quot;</td>
<td>match</td>
<td>no match</td>
<td></td>
</tr>
<tr>
<td>&quot;@&quot;</td>
<td>no match</td>
<td>match</td>
<td></td>
</tr>
<tr>
<td>&quot;95&quot;</td>
<td>no match</td>
<td>match</td>
<td></td>
</tr>
<tr>
<td>&quot;&quot;</td>
<td>no match</td>
<td>no match</td>
<td></td>
</tr>
<tr>
<td>&quot;)\n7</td>
<td>m&quot;</td>
<td>no match</td>
<td>no match</td>
</tr>
</tbody>
</table>

minDFA(R):

NFA(S-R):

Margus Veanes
program boolAssignmentDemo(t);
string s;
s := iter(c in t){b := false;}{
case ((c == 'a')):
    b := !(b) && b;
    b := b || b;
    b := !(b);
    yield (c);
    case (true):
        yield ('$');
};
return s;

// BEK says : boolAssignmentDemo is idempotent
// BEK says : boolAssignmentDemo is not reversible.

// The following JavaScript is equivalent to the BEK program:
function boolAssignmentDemo(t){
    var s =
    function ($){
        var result = new Array();
        for(i=0;i<$.$length; i++){
            var c =$.$[i];
            if ((c == String.fromCharCode(97))){
                b := (~b) && b;
                b := (b || b);
                b := ~(b);
                result.push(c);
            }
        }
    }

    !(c='a')/['$']
    !(c='a')/['$']
    (c='a')/['t']
    (!c='a')/['$']

Margus Veanes
David Molnar
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) {
        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;
}

void itoa(int n, char* s) {
    if (n < 0) {
        *s++ = '-';
        n = -n;
    }
    // Add digits to s
    ....
}
```

**Package:** java.util.Arrays

**Function:** binary_search

**Book:** Kernighan and Ritchie

**Function:** itoa (integer to ascii)

Analysis of millions of lines of Microsoft Code base
Modification in invariant checking

Switch to Z3 v2

Z3 v2 update

Switch to Boogie2

Attempt to improve Boogie/Z3 interaction

sat(and(F(k), and(T, not(next(P)))))
Scale: what is important - for applications?

Claim (as I see it):

- **Simplification** - lots of junk
- **Structural** - not random, (symmetry?)
- **Shallow** - unsat core
- **Repertoire** - cooperating methods
- **Decomposable** - solve simpler problems
- **Abstraction** - SAT < SMT

Are we there yet?

- Improve search methods and solvers,
- extend expressiveness, *tactics*,
- precise answers.
Scale: what is important - for applications?

Claim (as I see it):

- **Simplification** - lots of junk
- **Structural** - not random, (symmetry?)
- **Shallow** - unsat core
- **Repertoire** - cooperating methods
- **Decomposable** - solve simpler problems
- **Abstraction** - SAT < SMT

Are we there yet?

- **Improve search methods** and solvers,
- extend expressiveness, **tactics,**
- precise answers.
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) in a nutshell

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

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

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

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

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

Idea: DPLL(⊔)

Try branch \( a_1 \approx b_1 \wedge b_1 \approx a_2 \)
Implies \( a_1 \approx b_1 \approx a_2 \)
Collect implied equalities

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

Compute the \textit{join} \( \sqcup \) of the two equalities – common equalities are learned

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

Single case splits don’t suffice

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

Requires 2 case splits to collect implied equalities
**Method:** resolve literals in conflict clauses

Theorem (for EUF): \( \text{DPLL} + \text{CDER} + \text{Restart} \equiv_p \text{E-Resolution} \)

Informal Claim: \( \text{DPLL} + \text{CDTR} + \text{Restart} \equiv_p \text{Resolution} \)

**Practical?**

Method introduces extra literals (= junk)

\( \rightarrow \text{Throttle} \) resolution dynamically based on activity.
Eventually, many conflicts contain:

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

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:

$$a_1 \simeq b_1 \land b_1 \simeq a_2 \lor a_i \simeq c_i \land c_i \simeq a_{i+1}$$

Then DPLL(T) learns by itself:

$$a_1 \simeq b_1 \land b_1 \simeq a_2 \rightarrow a_1 \simeq a_2$$
Eventually, many conflicts contain:

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

Eventually, many conflicts contain:

\[ x_i \equiv u_i \land y_i \equiv u_i \quad u_i = v_0 \text{ or } u_i = v_1 \quad \text{for } i = 1 \ldots N \]

\[ \neg (f(x_N, \ldots, f(x_2, x_1) \ldots) \equiv f(y_N, \ldots, f(y_2, y_1) \ldots)) \]

Add:

\[ \left( \bigwedge_{i=1}^{N} x_i \equiv y_i \right) \rightarrow f(x_N, \ldots, f(x_2, x_1) \ldots) \equiv f(y_N, \ldots, f(y_2, y_1) \ldots) \]
Dynamic Ackermann Reduction

If *Congruence Rule* repeatedly learns

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

Then add clause for SAT core to use

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

Dynamic Ackermann Reduction with Transitivity

If *Equality Transitivity* repeatedly learns

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

Then add clause for SAT core to use

\[ u \approx v \land v \approx w \rightarrow v \approx w \]
Dynamic Ackermann Reduction

If **Congruence Rule** repeatedly learns

\[ f(v, v') \sim f(w, w') \] for literal \( f(v, v') \equiv f(w, w') \)

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

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

Then add clause for SAT core to use

\[ u \equiv v \land v \equiv w \rightarrow v \equiv w \]
\[ a < x_1 \land a < x_2 \land (x_1 < b \lor x_2 < b) \land
b < y_1 \land b < y_2 \land (y_1 < c \lor y_2 < c) \land
\]
\[ c < z_1 \land c < z_2 \land (z_1 < a \lor z_2 < a) \]
CDTR: Linear Difference Arithmetic

Top Two Most Active vertices

Add clause
\( a < x_1 < b \rightarrow a < b \)
Modern SMT solvers find resolution proofs
- unlike SAT solvers: SMT > \text{RES}_p
- Gap is real enough

Presented a technique for equalities
- Based on applying \textit{Resolution} to conflicts.
- \textbf{Dynamic} - to address literal introduction junk.

Just one of many possible optimizations.
- e.g. cutting plane proofs, arbitrary cuts (Frege)
- The devil is in the theory