Applications of SMT Solving at Microsoft

Nikolaj Bjørner
Microsoft Research
This Talk

- Using Decision Engines for Software @ Microsoft.
  - Dynamic Symbolic Execution
  - Bit-precise Scalable Static Analysis
  - and several others

- What is Important for Decision Engines
  - The sweet spot for SMT solvers
  - Shameless, blatant propaganda for the SMT solver Z3
A Decision Engine for Software

Some Microsoft engines:

- **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

They all use the SMT solver Z3.
.. Ok Z3 is not everything ..yet

Model Checker
For Multi-threaded Software
- k-bounded exhaustive
Cuzz:
- Randomized
The Inner Research Market @ MSFT
What is Z3?

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

Simplify
- OCaml
- SMT-LIB
- NET
- Native
- C
- F# quote

Quantifiers:
- E-matching
- Super-position

Model Generation:
- Finite Models
- Proof objects
- Assumption tracking

Parallel Z3

By Leonardo de Moura & Nikolaj Bjørner [http://research.microsoft.com/projects/z3]
Microsoft’s SMT solver Z3 is the snake oil when rubbed on solves all your problems.

Z3 Components:
- 9% SAT solver
- 14% Quantifier engine
- 10% Equality and functions
- 10% Arrays
- 20% Arithmetic
- 10% Bit-vectors
- …25% Secret Sauce
- ……2% Super Secret Sauce

Composition of snake oil

The composition of snake oil medicines varies markedly between products.

Snake oil sold in San Francisco’s Chinatown in 1989 was found \(^4\) to contain:

- 75% unidentified carrier material, including camphor
- 25% oil from Chinese water snakes, itself consisting of:
  - 20% eicosapentaenic acid (EPA) - an omega 3 derivative
  - 48% myristic acid (14:0)
  - 10% stearic acid (18:0)
  - 14% oleic acid (18:1ω9)
  - 7% linoleic acid (18:2ω6) plus arachidonic acid (20:4ω6)
Z3: Some Microsoft Clients

- .NET BCL
- Hyper-V
- Drivers
- PEX
- VCC
- SLAM/SDV

Is this path feasible?
Hoare Triples
Finite Program abstraction
Model
Proof
Engines for progressively succinct (first-order) frameworks
What is still *decidable*?
Encoding theories in less succinct frameworks.
*Efficiency* …
Encoding efficiently supported theories in less succinct frameworks.

What is still *decidable*?

Engines for progressively succinct (first-order) frameworks
What is SMT?
$x + 2 = y \Rightarrow f_{\text{read}}(\text{write}(a, x, 3), y - 2) = f(y - x + 1)$

Array Theory  Arithmetic  Uninterpreted Functions

$\text{read}(\text{write}(a, i, v), i) = v$

$i \neq j \Rightarrow \text{read}(\text{write}(a, i, v), j) = \text{read}(a, j)$
Domains from programs

- Bits and bytes: \( 0 = ((x - 1) \& x) \iff x = 00100000..00 \)
- Numbers: \( x + y = y + x \)
- Arrays: \( \text{read}(\text{write}(a, i, 4), i) = 4 \)
- Records: \( \text{mkpair}(x, y) = \text{mkpair}(z, u) \Rightarrow x = z \)
- Heaps: \( n \rightarrow^* n' \land m = \text{cons}(a, n) \Rightarrow m \rightarrow^* n' \)
- Data-types: \( \text{car}(\text{cons}(x, \text{nil})) = x \)
- Object inheritance: \( B <: A \land C <: B \Rightarrow C <: A \)
Application: Dynamic Symbolic Execution

- Pex, SAGE, Yogi, Vigilante
Dynamic Symbolic Execution

Run Test and Monitor

Execution Path

Path Condition

Test Inputs

Known Paths

Constraint System

Unexplored path

Seed

New input

Solve

Nikolai Tillmann, Peli de Halleux (Pex), Patrice Godefroid (SAGE)
Aditya Nori, Sriram Rajamani (Yogi), Jean Philippe Martin, Miguel Castro,
Manuel Costa, Lintao Zhang (Vigilante)
Internal user: “WEX Security team”

- Use 100s of dedicated machines 24/7 for months
- Apps: image processors, media players, file decoders,…
- Bugs: Write/read A/Vs, Crash,…
- Uncovered bugs not possible with “black-box” methods.
ABCDE: Application Beneficiary Challenge Direction Enabler

Enabler

FINITE MODEL GENERATION

Application
Dynamic
Symbolic
Execution

Direction
Model-guided
Dynamic
Symbolic Execution

Beneficiary

Using Template Models

Challenge

SAGE
Application:

**Bit-precise Scalable Static Analysis**

PREfix [Moy, B., Sielaff 2010]
What is wrong here?

```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

-INT_MIN = INT_MIN

3(INT_MAX+1)/4 + (INT_MAX+1)/4 = INT_MIN

Book: Kernighan and Ritchie
Function: itoa (integer to ascii)
C/C++ functions

int init_name(char **outname, uint n)
{
    if (n == 0) return 0;
    else if (n > UINT16_MAX) exit(1);
    else if (((*outname = malloc(n)) == NULL) {
        return 0xC0000095; // NT_STATUS_NO_MEM;
    }
    return 0;
}

int get_name(char* dst, uint size)
{
    char* name;
    int status = 0;
    status = init_name(&name, size);
    if (status != 0) {
        goto error;
    }
    strcpy(dst, name);
    error:
    return status;
}
iElement = m_nSize;
if( iElement >= m_nMaxSize )
{
    bool bSuccess = GrowBuffer( iElement+1 ),
    ...
}
::new( m_pData+iElement ) E( element );
m_nSize++;
ULONG AllocationSize;
while (CurrentBuffer != NULL) {
    if (NumberOfBuffers > MAX_ULONG / sizeof(MYBUFFER)) {
        return NULL;
    }
    NumberOfBuffers++;
    CurrentBuffer = CurrentBuffer->NextBuffer;
}
AllocationSize = sizeof(MYBUFFER)*NumberOfBuffers;
UserBuffersHead = malloc(AllocationSize);
Integration of Z3 into PREfix

A recent project with Yannick Moy.

😊: catches more bugs than old version of PREfix using incomplete ad-hoc solver.

😢: complete solver for bit-vector operations incurs overhead compared to incomplete solver.

Ran v1 through “large Microsoft code-base”

Filed a few dozen bugs during the first run.
Application: Program Verification
- Spec#, VCC, HAVOC
Extended Static Checking and Verification

Hyper-V

VCC

HAVOC

Boogie

Win. Modules

Bug path

Z3

F7/FINE

Rustan Leino, Mike Barnet, Michał Moskal, Shaz Qadeer, Shuvendu Lahiri, Herman Venter, Wolfram Schulte, Ernie Cohen, Khatib Braghaven, Cedric Fournet, Andy Gordon, Nikhil Swamy
#include <vcc2.h>

typedef struct _BITMAP {
    UINT32 Size;    // Number of bits ...
    PUINT32 Buffer; // Memory to store ...
}

// private invariants
invariant(Size > 0 && Size % 32 == 0)

Verification Condition Generator

Annotated C

Boogie

http://vcc.codeplex.com/
Using Z3’s support for quantifier instantiation + theories
Modification in invariant checking

Switch to Z3 v2

Z3 v2 update

Switch to Boogie2

Attempt to improve Boogie/Z3 interaction

VCC Performance Trends Nov 08 – Mar 09

Viel Spaß und liebe Grüße an Lieven,
Markus
Application: Model-Based Design

- FORMULA
Use Design Space Exploration to identify valid candidate architectures
FORMULA: Diversified Search

Subtract all isomorphic solutions

SMT Formula

Z3 Solver

Diversify and Constrain Search Space

Remember this model
Enabler

GENERATING FINITE MODELS

Application
Model-Based Design

Direction
Embedded Real-time systems

Challenge
Quantifier Elimination

ABCDE
Application: Model-Based Testing
- SpecExplorer, M3
Example Microsoft protocol:
- SMB2 (= remote file) Protocol Specification
- 200+ other Microsoft Protocols

Tools:
Symbolic Exploration of protocol models to generate tests.

Pair-wise independent input generation for constrained algebraic data-types.

Design time model debugging using
- Bounded Model Checking
- Bounded Conformance Checking
- Bounded Input-Output Model Programs
Next steps – Model-based Testing

Enabler

SEARCH ONLY
RELEVANT SPACE

Application
Model-based Testing

Direction
Program Synthesis

SEARCH STRATEGIES

Challenge
Selected Z3 Technologies
Model-based Theory Combination

**Foundations**
- 1979 Nelson, Oppen - Framework
- 1996 Tinelli & Harindi. N.O Fix
- 2000 Barrett et.al N.O + Rewriting
- 2002 Zarba & Manna. “Nice” Theories
- 2004 Ghilardi et.al. N.O. Generalized

**Efficiency using rewriting**
- 1984 Shostak. Theory solvers
- 1996 Cyrluk et.al Shostak Fix #1
- 1998 B. Shostak with Constraints
- 2001 Rueß & Shankar Shostak Fix #2
- 2004 Ranise et.al. N.O + Superposition

2001: Moskewicz et.al. Efficient DPLL made guessing cheap

2006 Bruttomesso et.al. Delayed Theory Combination

**2007 de Moura & B. Model-based Theory Combination**

2010 Jovanovic & Barrett. Sharing is Caring
A basis of operations

\[ \text{write}(a, i, v) = \lambda j. \text{ite}(i = j, v, a[j]) \]

\[ K(v) = \lambda j. v \]

\[ \text{map}_f (a, b) = \lambda j. f (a[j], b[j]) \]

\[ \delta(a) = a[\varepsilon(a)] \]
## Derived operations

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \emptyset )</td>
<td>( K(\text{false}) )</td>
</tr>
<tr>
<td>{a}</td>
<td>( \text{write}(\emptyset, a, \text{true}) )</td>
</tr>
<tr>
<td>( a \in A )</td>
<td>( A[a] )</td>
</tr>
<tr>
<td>( A \cup B )</td>
<td>( \text{map}_\vee(A, B) )</td>
</tr>
<tr>
<td>( A \cap B )</td>
<td>( \text{map}_\wedge(A, B) )</td>
</tr>
<tr>
<td>finite(A)</td>
<td>( (\delta(A) = \text{false}) )</td>
</tr>
<tr>
<td>( \emptyset_{\text{Bag}} )</td>
<td>( K(0) )</td>
</tr>
<tr>
<td>{a}</td>
<td>( \text{write}(\emptyset, a, 1) )</td>
</tr>
<tr>
<td>( \text{mult}(a, A) )</td>
<td>( A[a] )</td>
</tr>
<tr>
<td>( A \oplus B )</td>
<td>( \text{map}_+ (A, B) )</td>
</tr>
<tr>
<td>( A \sqcap B )</td>
<td>( \text{map}_{\text{min}}(A, B) )</td>
</tr>
<tr>
<td>finite_{\text{Bag}}(A)</td>
<td>( (\delta(A) = 0) )</td>
</tr>
</tbody>
</table>
Match: $\text{read}(\text{write}(A,I,V),I) = \text{read}(\text{write}(a,g(c),c),f(d,a))$

Assuming

$E = \{ g(a) = f(b, c), b = d, a = c \}$

Efficiency through:

- **Code trees:**
  Runtime program specialization.

- **Inverted path indexing:**
  When new equality enters, walk from sub-terms upwards to roots in index.

[CADE 2007]
Efficient E-graph Matching

Match: \( \text{read(write}(A,l,V),l) = \text{read(write}(a,g(c),c),f(b,a)) \)

Assuming
\[
E = \{ \ g(a) = f(b, c), \ b = d, \ a = c \ \}
\]

Efficiency through:

- **Code trees:**
  Runtime program specialization.

- **Inverted path indexing:**
  When new equality enters, walk from sub-terms upwards to roots in index.

[CADE 2007]
Efficient E-graph Matching

Match: \( \text{read(write}(A,I,V),I) = \text{read(write}(a,g(c),c),f(b,c)) \)

Assuming

\[ E = \{ \ g(a) = f(b, c), b = d, a = c \ \} \]

Efficiency through:

- **Code trees:**
  Runtime program specialization.

- **Inverted path indexing:**
  When new equality enters, walk from subterms upwards to roots in index.

[CADE 2007]
Efficient E-graph Matching

Match: $\text{read}(\text{write}(A, I, V), I) = \text{read}(\text{write}(a, g(c), c), g(a))$

Assuming

$E = \{ g(a) = f(b, c), b = d, a = c \}$

Efficiency through:

- **Code trees:**
  Runtime program specialization.

- **Inverted path indexing:**
  When new equality enters, walk from subterms upwards to roots in index.
Efficient E-graph Matching

Match: \( read(write(A,I,V),I) = read(write(a,g(c),c),g(c)) \)
Assuming
\[ E = \{ \ g(a) = f(b, c), \ b = d, \ a = c \ \} \]

Efficiency through:
- **Code trees:**
  Runtime program specialization.
- **Inverted path indexing:**
  When new equality enters, walk from sub-terms upwards to roots in index.

[CADE 2007]
Linear quantifier Elimination as an Abstract Decision Procedure

- SMT for QE has some appeal:
  - Just use SMT(LA/LIA) for closed formulas.

Algorithms:

[Diagram with algorithms: Fourier Motzkin → Resolution → Omega Test 
 Loos-Weispfenning → Case split+ Virtual subst → Cooper 
 Abstract Decision Proc → Case split+ Resolution → Abstract Decision Proc]

[IJCAR 2010]
SMT solvers are a great fit for software tools

Current main applications:
- Test-case generation.
- Verifying compilers.
- Model Checking & Predicate Abstraction.
- Model-based testing and development

Future opportunities in SMT research and applications abound