Modern Satisfiability Modulo Theories Solvers in Program Analysis
Lectures

Wednesday 10:45–12:15
An Introduction to Z3 with Applications

Thursday August 30\textsuperscript{th} 15:45–17:15
Introduction to SAT and SMT

Friday 10:30–10:45
Theories and Solving Algorithms

Friday 15:45–17:15
Advanced & Summary:
Quantifiers, Arrays, Fixed-points
Plan

I. Introduction to Z3 – An Efficient SMT Solver
   I. Architecture
   II. Interaction – Input formats
   III. Theories

II. Introduction to Applications
   I. Basic examples of SMT solving
   II. A selection of applications using Z3/SMT.
Takeaways:

• Many program analysis, verification and test tools solve problems that can be reduced to logical formulas and transformations between logical formulas at their core.

• Modern SMT solvers are a often good fit.
  – Handle domains found in programs directly.

• The selected examples are intended to show instances where sub-tasks are reduced to SMT/Z3.
Z3 – AN EFFICIENT SMT SOLVER

http://research.microsoft.com/projects/z3
What is Z3?

Simplify

SMT-LIB

Native

Theories

Bit-Vectors

Arrays

Lin-arithmetic

Nonlin-arithmetic

Recursive Datatypes

Floating points

Free (uninterpreted) functions

Quantifiers:

E-matching

Super-position

Tactics

Model Generation:

Finite/parametric

Proof objects

Assumption tracking

By Leonardo de Moura, Nikolaj Bjørner, Christoph Wintersteiger
Z3 is Cutting Edge

http://smtcomp.org

**Summary View**

The job selection includes SMT-COMP 2011, the official SMT-COMP’11 competition run

<table>
<thead>
<tr>
<th>Solver</th>
<th>Score</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z3</td>
<td>201 / 205</td>
<td></td>
</tr>
<tr>
<td>MathSAT5</td>
<td>199 / 205</td>
<td></td>
</tr>
<tr>
<td>CVC4 1.0rc1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>veriT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>opensmt</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SMT2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CVC3 v2.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MathSAT 5, 2010 winner</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Z3 is Free Software for research

http://smtcomp.org
Z3: Little Engines of Proof

Freely available from http://research.microsoft.com/projects/z3
Input Formats

• Text:
  – SMT-LIB2 - main exchange format for SMT solvers
  – Log - low-level log for replay
  – Datalog engine - a Datalog format for the fixed-point engine
  – DIMACS - a format for propositional SAT

• Programmatic:
  – C - API functions exposed for C
  – Ocaml - Ocaml wrapper around C API
  – .NET - .NET wrapper around C API
  – Python - Try it on http://rise4fun.com/z3py
  – Scala - by Phillipe Suter
A Primer on SMT-LIB2

Online interactive tutorial

http://rise4fun.com/z3/tutorial
THEORIES
Theories

Uninterpreted functions

```
(z3)
Is this formula satisfiable? Ask z3!
1 (declare-sort () A)
2 (declare-fun f (A) A)
3 (declare-const a A)
4 (assert (= a (f (f a))))
5 (assert (= a (f (f (f a)))))
6 (check-sat)
7 (get-model)
8 (echo "Adding contradiction")
9 (assert (not (= a (f a))))
10 (check-sat)
```
Theories

Uninterpreted functions

Arithmetic (linear)
Theories

Uninterpreted functions

Arithmetic (linear)

Bit-vectors
Theories

Uninterpreted functions
Arithmetic (linear)
Bit-vectors
Algebraic data-types

Is this formula satisfiable? Ask z3!

1 ; This example illustrates the declaration and
2 ; use of recursive datatypes in Z3.
3
4 ; List is a builtin datatype defined as in the SMT
5 (declare-fun l1 () (List Int))
6 (declare-fun l2 () (List Int))
7 (declare-fun l3 () (List Int))
8 (declare-fun x () Int)
9 (declare-fun y () Int)
10 (assert (not (= 11 nil)))
11 (assert (not (= 12 nil)))
12 (push)
13 (assert (= (head l1) (head l2)))
14 (assert (= (tail l1) (tail l2)))
15 (push)
16 (assert (not (= 11 12)))
17 (check-sat)
18 (pop)
19 (check-sat)
20 (eval l1)
21 (eval l2)

 unsat
 sat
 (insert 0 nil)
 (insert 0 nil)
 sat
Theories

Uninterpreted functions
Arithmetic (linear)
Bit-vectors
Algebraic data-types
Arrays
Theories

Uninterpreted functions
Arithmetic (linear)
Bit-vectors
Algebraic data-types
Arrays

Polynomial Arithmetic
Theories

Uninterpreted function

Arithmetic (linear)

Bit-vectors

Algebraic data-types

Arrays

Polynomial Arithmetic
USER-INTERACTION AND GUIDANCE
(define-sorts ((A (Array Int Int)))))
(declare-funs ((x Int) (y Int) (z Int)))
(declare-funs ((a1 A) (a2 A) (a3 A)))
(assert (= (select a1 x) x))
(assert (= (store a1 x y) a1))
(check-sat)
(get-info model)

Is this SMT formula satisfiable? Click 'ask Z3'! Read more or watch the video.

sat
("model" 
(define x 0)
(define a1 as-array[k!0])
(define y 0)
(define (k!0 (x1 Int))
(if (= x1 0) 0 1))")
Logical Formula

Unsat/Proof

Logical Formula

Unsat/Proof

Logical Formula

Unsat/Proof
Simplify Logical Formula

(declare-fun x () Real)
(declare-fun y () Real)
(simplify (>= x (+ x y)))

Is this SMT formula satisfiable? Click 'ask Z3'! Read more or watch the video.

(<= y 0.0)
Implied Equalities

- $x$ and $y$ are equal
- $z + y$ and $x + z$ are equal

Logical Formula

$$(\text{declare-funs ((x Int) (y Int) (z Int)))}
(\text{assert (>= x y)})
(\text{assert (>= z x)})
(\text{assert (>= y z)})
(\text{get-implied-equalities}
\begin{align*}
x + z & \equiv y + x + z 
\end{align*})$$
Interaction

(set-option set-param "ELIM_QUANTIFIERS" "true")
declare-funs ((a Int) (b Int))
simplify (forall (y Int) (or (<= a y) (> b y)))

ask z3
Is this SMT formula satisfiable? Click 'ask Z3'! Read more or watch the video.

(let ((?x34 (* (~ 1) b)))
(let ((?x145 (+ a ?x34)))
(let ((?x146 (>= ?x145 1)))
(not ?x146))))
Is this SMT formula satisfiable?

Click 'ask Z3'! Read more or watch the video.

Unsat

((or p q)
(=> r s)
(or r p)
(or r s)
(not (and r q))
(not (and s p)))
APPLICATIONS OF 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**: 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.
Applications

- Test case generation
- Verifying Compilers
- Predicate Abstraction
- Invariant Generation
- Type Checking
- Model Based Testing
A formula $F$ is valid
Iff
$\neg F$ is unsatisfiable
A formula $F$ is valid Iff

$\neg F$ is unsatisfiable

Theorem Prover/Satisfiability Checker

Timeout

Memout

Satisfiable Model

Unsatisfiable Proof
Verification/Analysis Tool: “Template”

Problem

Verification/Analysis Tool

Logical Formula

Theorem Prover/Satisfiability Checker

Satisfiable (Counter-example)

Unsatisfiable
SMT EXAMPLE

Basic
Is formula $\varphi$ satisfiable modulo theory $T$?

SMT solvers have specialized algorithms for $T$. 

SMT solvers have specialized algorithms for $T$. 

Is formula $\varphi$ satisfiable modulo theory $T$?
Satisfiability Modulo Theories (SMT)

\[ x + 2 = y \Rightarrow f(\text{select}(\text{store}(a, x, 3), y - 2)) = f(y - x + 1) \]

Array Theory | Arithmetic | Uninterpreted Functions

\[ \text{select}(\text{store}(a, i, v), i) = v \]
\[ i \neq j \Rightarrow \text{select}(\text{store}(a, i, v), j) = \text{select}(a, j) \]
SMT EXAMPLE

Job Shop Scheduling
Job Shop Scheduling

\[ \zeta(s) = 0 \Rightarrow s = \frac{1}{2} + ir \]
Job Shop Scheduling

Constraints:

**Precedence:** between two tasks of the same job

**Resource:** Machines execute at most one job at a time

\[ [start_{2,2}..end_{2,2}] \cap [start_{4,2}..end_{4,2}] = \emptyset \]
Job Shop Scheduling

Constraints:

Precedence:

Encoding:

$t_{2,3}$ - start time of job 2 on mach 3

$d_{2,3}$ - duration of job 2 on mach 3

$t_{2,3} + d_{2,3} \leq t_{2,4}$

Resource:

$[\text{start}_{2,2}..\text{end}_{2,2}] \cap [\text{start}_{4,2}..\text{end}_{4,2}] = \emptyset$

$t_{2,2} + d_{2,2} \leq t_{4,2}$

$\lor$

$t_{4,2} + d_{4,2} \leq t_{2,2}$

Not convex
Job Shop Scheduling

<table>
<thead>
<tr>
<th>( d_{i,j} )</th>
<th>Machine 1</th>
<th>Machine 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job 1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Job 2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Job 3</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

\[ \text{max} = 8 \]

Solution
\( t_{1,1} = 5, \ t_{1,2} = 7, \ t_{2,1} = 2, \)
\( t_{2,2} = 6, \ t_{3,1} = 0, \ t_{3,2} = 3 \)

Encoding
\[
\begin{align*}
(t_{1,1} \geq 0) \land (t_{1,2} \geq t_{1,1} + 2) \land (t_{1,2} + 1 \leq 8) \land \\
(t_{2,1} \geq 0) \land (t_{2,2} \geq t_{2,1} + 3) \land (t_{2,2} + 1 \leq 8) \land \\
(t_{3,1} \geq 0) \land (t_{3,2} \geq t_{3,1} + 2) \land (t_{3,2} + 3 \leq 8) \land \\
((t_{1,1} \geq t_{2,1} + 3) \lor (t_{2,1} \geq t_{1,1} + 2)) \land \\
((t_{1,1} \geq t_{3,1} + 2) \lor (t_{3,1} \geq t_{1,1} + 2)) \land \\
((t_{2,1} \geq t_{3,1} + 2) \lor (t_{3,1} \geq t_{2,1} + 3)) \land \\
((t_{1,2} \geq t_{2,2} + 1) \lor (t_{2,2} \geq t_{1,2} + 1)) \land \\
((t_{1,2} \geq t_{3,2} + 3) \lor (t_{3,2} \geq t_{1,2} + 1)) \land \\
((t_{2,2} \geq t_{3,2} + 3) \lor (t_{3,2} \geq t_{2,2} + 1))
\end{align*}
\]
Job Shop Scheduling

\[(t_{1,1} \geq 0) \land (t_{1,2} \geq t_{1,1} + 2) \land (t_{1,2} + 1 \leq 8) \land (t_{2,1} \geq 0) \land (t_{2,2} \geq t_{2,1} + 3) \land (t_{2,2} + 1 \leq 8) \land (t_{3,1} \geq 0) \land (t_{3,2} \geq t_{3,1} + 2) \land (t_{3,2} + 3 \leq 8) \land ((t_{1,1} \geq t_{2,1} + 3) \lor (t_{2,1} \geq t_{1,1} + 2)) \land ((t_{1,1} \geq t_{3,1} + 2) \lor (t_{3,1} \geq t_{1,1} + 2)) \land ((t_{2,1} \geq t_{3,1} + 2) \lor (t_{3,1} \geq t_{2,1} + 3)) \land ((t_{1,2} \geq t_{2,2} + 1) \lor (t_{2,2} \geq t_{1,2} + 1)) \land ((t_{1,2} \geq t_{3,2} + 3) \lor (t_{3,2} \geq t_{1,2} + 1)) \land ((t_{2,2} \geq t_{3,2} + 3) \lor (t_{3,2} \geq t_{2,2} + 1))\]

Efficient solvers:
- Floyd-Warshall algorithm
- Ford-Fulkerson algorithm

\[
z - t_{1,1} \leq 0
\]
\[
z - t_{2,1} \leq 0
\]
\[
z - t_{3,1} \leq 0
\]
\[
t_{3,2} - z \leq 5
\]
\[
t_{3,1} - t_{3,2} \leq -2
\]
\[
t_{2,1} - t_{3,1} \leq -3
\]
\[
t_{1,1} - t_{2,1} \leq -2
\]

\[
z - z = 5 - 2 - 3 - 2 = -2 < 0
\]
EXAMPLE EXERCISES

Follow-on to lectures by Hoare and Petersen
Basic Arithmetic

\[ x \ominus y \triangleq \max(0, (x - y)) \]

True or false?
\[(x \ominus a) + a \leq x,\]

SMT-LIB2 formulation:

```lisp
(define-fun Max ((x Int) (y Int)) Int (if (> x y) x y))
(define-fun ominus ((x Int) (y Int)) Int (Max 0 (- x y)))
(declare-const x Int)
(declare-const a Int)
(assert (not (<= (+ (ominus x a) a) x)))
(check-sat)
(get-model)
```
Is this formula satisfiable? Ask z3!

(declare-fun meet (A A) A)
(declare-fun join (A A) A)
(declare-fun semidistributivity (A B C) (iff (and ((meet (meet a b) c) (meet a (meet b c))) (meet (meet a c) (meet a b c)))))

Theorems about wp

\[ P; Q \Rightarrow R \iff P \Rightarrow wp(Q, R) \]
\[ P \Rightarrow wp(Q, P; Q) \]
\[ wp(Q, R); Q \Rightarrow R \]
\[ wp(Q_1 \lor Q_2, R) = wp(Q_1, R) \land wp(Q_2, R) \]
\[ wp(Q, R_1 \land R_2) = wp(Q, R_1) \land wp(Q, R_2) \]
\[ wp(Q_1; Q_2, R) = wp(Q_1, wp(Q_2, R)) \]
\[ wp(Q_1, R_1) \land wp(Q_2, R_2) \Rightarrow wp(Q_1 \land Q_2, R_1 \land R_2) \]

(adjoint) (adjoint) (adjoint) (choice) (conjunctivity) (composition) (exchange)
Is this formula satisfiable? Ask z3!

1 (declare-sort A)
2 (declare-sort B)
3 (declare-fun f (A) B)
4 (declare-fun g (B) A)
5 (declare-fun RA (A A) Bool)
6 (declare-fun RB (B B) Bool)
7
8
9 (assert (forall ((a1 A) (a2 A) (a3 A)))
   (⇒ (and (RA a1 a2) (RA a2 a3)) (RA a1 a3)))
10
11
12 (assert (forall ((b1 B) (b2 B) (b3 B)))
13   (⇒ (and (RB b1 b2) (RB b2 b3)) (RB b1 b3)))
14
15
16 (define-fun adj1 () Bool
17   (forall ((a A) (b B)) (= (RB (f a) b) (RA a (g b)))))
18
19
20 (define-fun adj2 () Bool
21   (= (forall ((b B)) (RB (f (g b)) b))
    home  tutorial  video  permalink
Follow on Exercises

Prove the other properties for WP

Axioms proved from calculi

from Hoare
- \( p ; (q \lor r) \Rightarrow p ; q \lor p ; r \)
- \( p ; r \lor q ; r \Rightarrow (p \lor q) ; r \)

from Milner
- \( (p \lor q) ; r \Rightarrow (p ; r) \lor (q ; r) \)
- \( p ; q \lor p ; r \Rightarrow p ; (q \lor r) \)

from both
- \( p ; (q ; r) \Rightarrow (p ; q) ; r \)
- \( (p ; q) ; r \Rightarrow p ; (q ; r) \)
- exchange law

Prove Axioms for Hoare and Milner calculi
SMT APPLICATION

Test case generation
Test generation tools

SAGE

Internal. For Security Fuzzing

Runs on x86 instructions

External. For Developers

Runs on .NET code

Try it on: http://pex4fun.com

Finding security bugs before the black hat
SMT APPLICATION

Static Analysis

Integrating Z3 with PREfix static analyzer. Yannick Moy, David Selaff, B
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;
}

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)
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;
}
```c
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;
}
```

---

**C/C++ functions**

**model for function init_name**

outcome init_name_0:
  - guards: n == 0
  - results: result == 0
outcome init_name_1:
  - guards: n > 0; n <= 65535
  - results: result == 0xC0000095
outcome init_name_2:
  - guards: n > 0; n <= 65535
  - constraints: valid(outname)
  - results: result == 0; init(*outname)
C/C++ functions

model for function init_name
outcome init_name_0:
  guards: n == 0
  results: result == 0
outcome init_name_1:
  guards: n > 0; n <= 65535
  results: result == 0xC0000095
outcome init_name_2:
  guards: n > 0; n <= 65535
  constraints: valid(outname)
  results: result == 0; init(*outname)

path for function get_name
  guards: size == 0
  constraints:
    facts: init(dst); init(size); status == 0

pre-condition for function strcpy
  init(dst) and valid(name)

warnings

The PREfix Static Analysis Engine

```
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);
LONG l_sub(LONG l_var1, LONG l_var2)
{
    LONG l_diff = l_var1 - l_var2; // perform subtraction
    // check for overflow
    if ( (l_var1>0) && (l_var2<0) && (l_diff<0) ) l_diff=0x7FFFFFFF
}
Overflow on unsigned addition

for (uint16 uID = 0; uID < uDevCount && SUCCEEDED(hr); uID++) {
    ...
    if (SUCCEEDED(hr)) {
        uID = uDevCount; // Terminates the

    Possible overflow
    uID == UINT_MAX

    Loop does not terminate
DWORD dwAlloc;
dwAlloc = MyList->nElements * sizeof(MY_INFO);
if(dwAlloc < MyList->nElements)  
    ...  // return
MyList->pInfo = malloc(dwAlloc);
Bit-precise analysis

Concatenation

\[
\begin{pmatrix}
1 & 0 & 1 & 0 & 1 & 1 \\
\end{pmatrix}
\times
\begin{pmatrix}
0 & 1 & 1 & 0 & 0 & 1 \\
\end{pmatrix}
= 
\begin{pmatrix}
1 & 0 & 1 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 & 0 & 1 \\
\end{pmatrix}
\]

Extraction

\[
\begin{pmatrix}
1 & 0 & 1 & 0 & 1 & 1 \\
\end{pmatrix}
[4:2] = 
\begin{pmatrix}
0 & 1 & 0 \\
\end{pmatrix}
\]

Bit-wise and

\[
\begin{pmatrix}
1 & 0 & 1 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 & 0 & 1 \\
\end{pmatrix}
\wedge
= 
\begin{pmatrix}
0 & 0 & 1 & 0 & 0 & 1 \\
\end{pmatrix}
\]

Addition

\[
\begin{pmatrix}
1 & 0 & 1 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 & 0 & 1 \\
\end{pmatrix}
+ 
= 
\begin{pmatrix}
0 & 0 & 0 & 1 & 0 & 0 \\
\end{pmatrix}
\]

Modular arithmetic
Bit-precise analysis

Method: Encode bit-vectors bit-by-bit into Boolean Satisfiability

\[
\begin{align*}
\text{Multiplication} & \quad [\text{out}_3, \text{out}_2, \text{out}_1, \text{out}_0] = [a_3, a_2, a_1, a_0] \times [b_3, b_2, b_1, b_0] \\
\text{O(n^2) clauses} & \quad \text{SAT solving time increases exponentially. Similar for BDDs.} \\
\text{Brute-force enumeration + evaluation faster for 20 bits.}
\end{align*}
\]
SMT APPLICATION

Verified Software
Microsoft Verifying C Compiler

```
#include <vcc2.h>

typedef struct _BITMAP {
    UINT32 Size; // Number of bits
    UINT32 Buffer; // Memory to store
    // private invariants
    invariant(Size > 0 && Size % 32 == 0)
    ...

:assumption
    (forall (?x Int) (?y Int)
    (iff
        (= (IntEqual ?x ?y) boolTrue
        (=?x ?y)))
    :formula
    (flatten
        ...
```

Partners:
• European Microsoft Innovation Center
• Microsoft Research
• Microsoft’s Windows Division
• Universität des Saarlandes

co-funded by the German Ministry of Education and Research

http://www.verisoftxt.de
Hypervisor: Scale & Speed

- VCs have several Mega-bytes
- Thousands of non ground formulas
- Developers are willing to wait at most 5 min per VC

Are you willing to wait more than 5 min for your compiler?
Verification Attempt Time vs. Satisfaction and Productivity

- Loose Focus
- Loose Interest
Why did my proof attempt fail?

1. My annotations are not strong enough!
weak loop invariants and/or contracts

2. My theorem prover is not strong (or fast) enough.
Send “angry” email to Nikolaj and Leo.
Attempt to improve Boogie/Z3 interaction
Modification in invariant checking
Switch to Boogie2
Switch to Z3 v2
Z3 v2 update
The Importance of Speed

Subject: FW: Der neue Z3 ist höllisch schnell (und ich meine kein Auto)

Fyi.


Viel Spaß und liebe Grüße an Lieven,
Markus
Building Verve

Source file
- Nucleus.bpl (x86)
- Kernel.obj (x86)

Verification tool
- Boogie/Z3
- TAL checker

Compilation tool
- Translator/Assembler
- Linker/ISO generator

C# compiler
- Kernel.cs

9 person-months

Safe to the Last Instruction / Jean Yang & Chris Hawbliztl PLDI 2010
SMT APPLICATION

Formal Language Theory and Security
Why string analysis?
(motivating scenario)

req = http://www.x.com/%c0%ae%c0%ae/%c0%ae%c0%ae/private/

Analysis question: Does utf8decode reject overlong utf8-encodings such as "%C0%AE" for "."?

access granted to "../..private/"

1) security check: req must not contain "../"
2) dir = utf8decode("%c0%ae%c0%ae%c0%ae/private/") = "../..private/

Relativized Formal Language Theory

Symbolic Word Transducers
≈
Classical Word Transducers \textit{modulo} \(Th(\Sigma)\)

Classical Word Transducers
(e.g. decoding automata, rational transductions)

Classical I/O Automata
(e.g. Mealy machine)

Symbolic Word Acceptors
≈
Classical Word Acceptors \textit{modulo} \(Th(\Sigma)\)

(Classical Word Acceptors (NFA, DFA))

String transformation

Regex matching
Rex & Bek – Symbolic RegEx & Transducers

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 S</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;88i&quot;</td>
<td>match</td>
<td>match</td>
<td></td>
</tr>
<tr>
<td>&quot;\n&quot;</td>
<td>match</td>
<td>no match</td>
<td>Your match is different from the secret regex</td>
</tr>
<tr>
<td>\n\n</td>
<td>match</td>
<td>no match</td>
<td>Your match is different from the secret regex</td>
</tr>
<tr>
<td>&quot;0&quot;</td>
<td>no match</td>
<td>match</td>
<td>Your match is different from the secret regex</td>
</tr>
<tr>
<td>&quot;55&quot;</td>
<td>no match</td>
<td>match</td>
<td>Your match is different from the secret regex</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
Symbolic Finite Transducer (SFT)

• Classical transducer *modulo* a rich *label theory*
• Core Idea: represent labels with guarded transformation functions
  – Separation of concerns: finite graph / theory of labels

Concrete transitions:

```
p \rightarrow [p, 80_{16} \leq x \leq 7FF_{16} | \langle x_{(10,6)}, 80_{16} | x_{(5,0)} \rangle]
```

Symbolic transition:

```
\lambda x. \text{guard bitvector operations}
q \rightarrow [q, 80_{16} \leq x \leq 7FF_{16}]
```
SMT APPLICATION

Software Model Checking
Static Driver Verifier

**SLAM** – part of SDV (Windows 7, 8)
Z3 is used for:
  - Predicate abstraction
  - Counter-example refinement

**Yogi** – next generation SDV
Z3 is used for:
  - Refining transition abstractions

**Corral** – uses reachability modulo theories

**SLAyer** – checks memory safety using separation logic
Z3 & Static Driver Verifier: SLAM

• All-SAT
  – Better (more precise) Predicate Abstraction
• Unsatisfiable cores
  – Why the abstract path is not feasible?
  – Fast Predicate Abstraction
Summary

Introduction to Z3: An Efficient SMT Solver

Introduction to Applications:

- Exemplary applications
- Tools built around Z3

Tomorrow. More technical:

- Introduction to SAT solving.
- Introduction to SMT solving.