Applications of SMT Solving at Microsoft

Nikolaj Bjørner
Microsoft Research

FSE & RiSE

This Talk

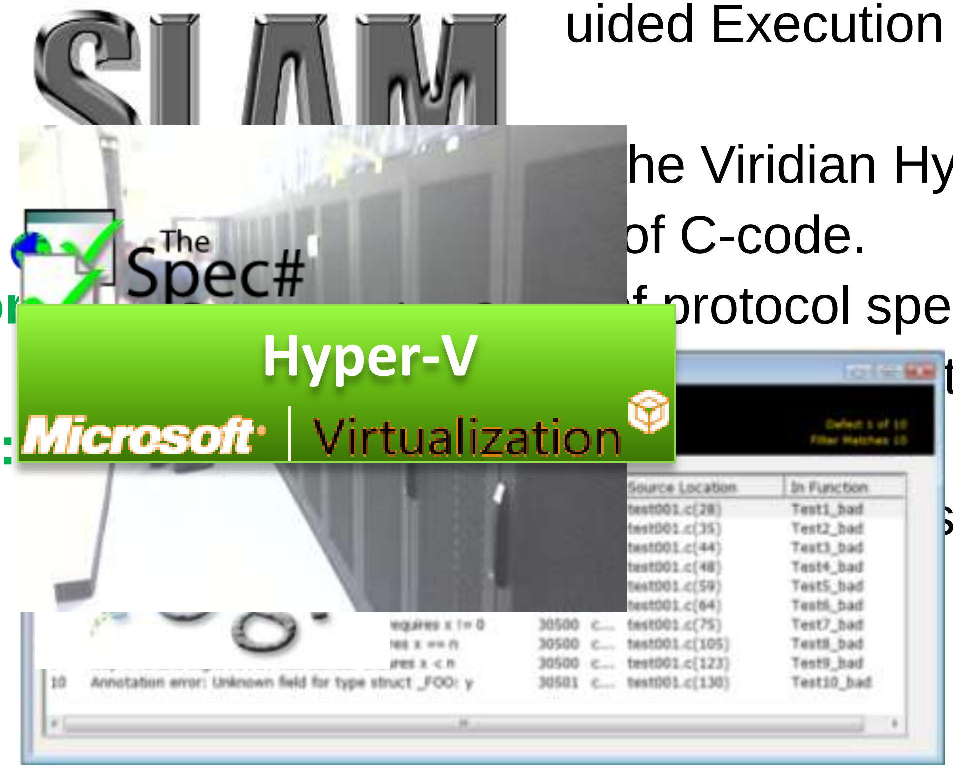
- Using Decision Engines for Software @ Microsoft.
  - Dynamic Symbolic Execution
  - Bit-precise Scalable Static Analysis
  - and several others

- 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:** uided Execution
- **Spec#:**
- **VCC:** he Viridian Hyper-Visor
- **HAVOC:** of C-code.
- **SpecExplor** f protocol specs.
- **Yogi:** tion.
- **FORMULA:**
- **F7:** s
- **M3:**
- **VS3:**



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?

Simplify
SMT-LIB
Native

Theories

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

OCaml
.NET
C
F# quote

Quantifiers: E-matching

Model Generation: Finite Models

Quantifiers: Super-position

Proof objects

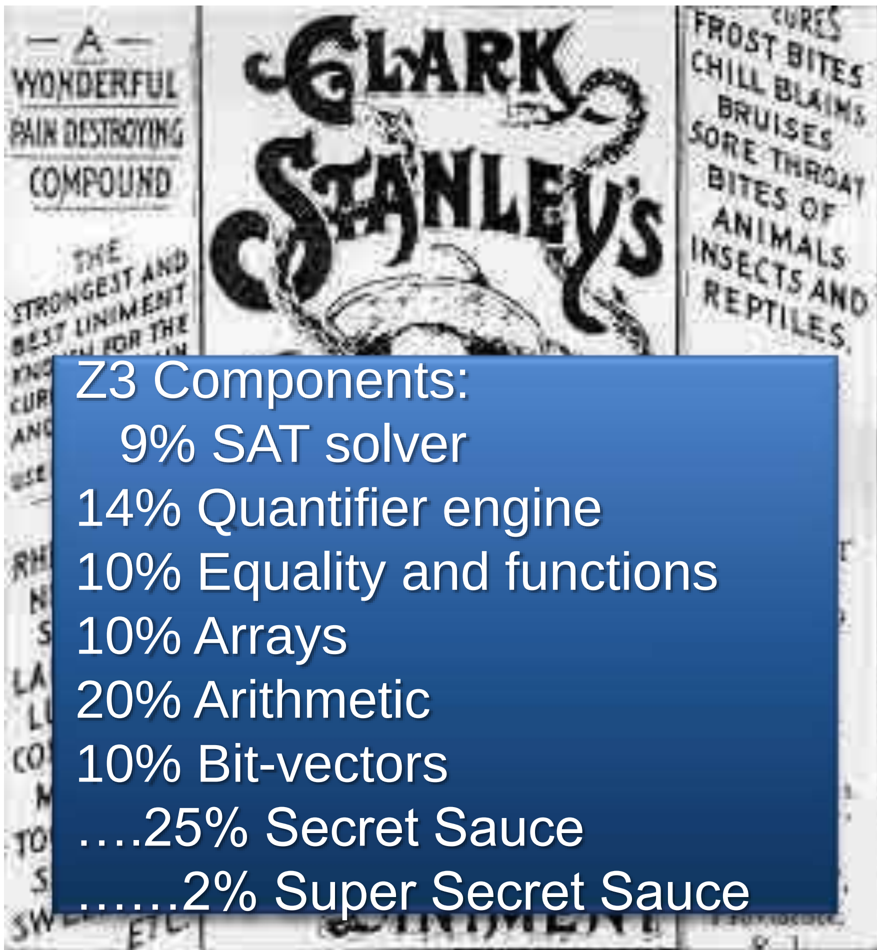
Parallel Z3

Assumption tracking

By Leonardo de Moura & Nikolaj Bjørner http://research.microsoft.com/projects/z3

Message ☺

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

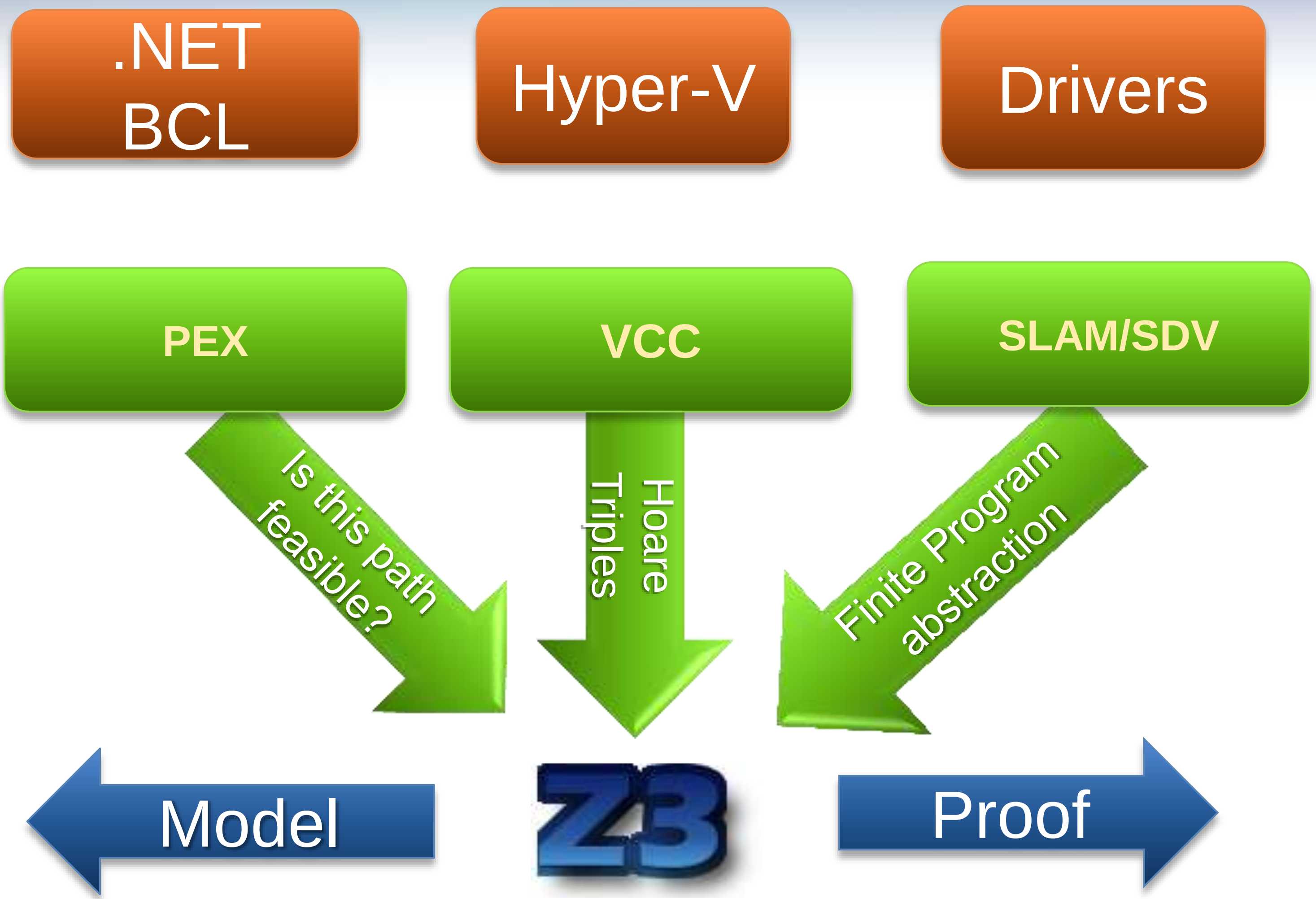
The composition of snake oil medicines varies markedly between products.

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)

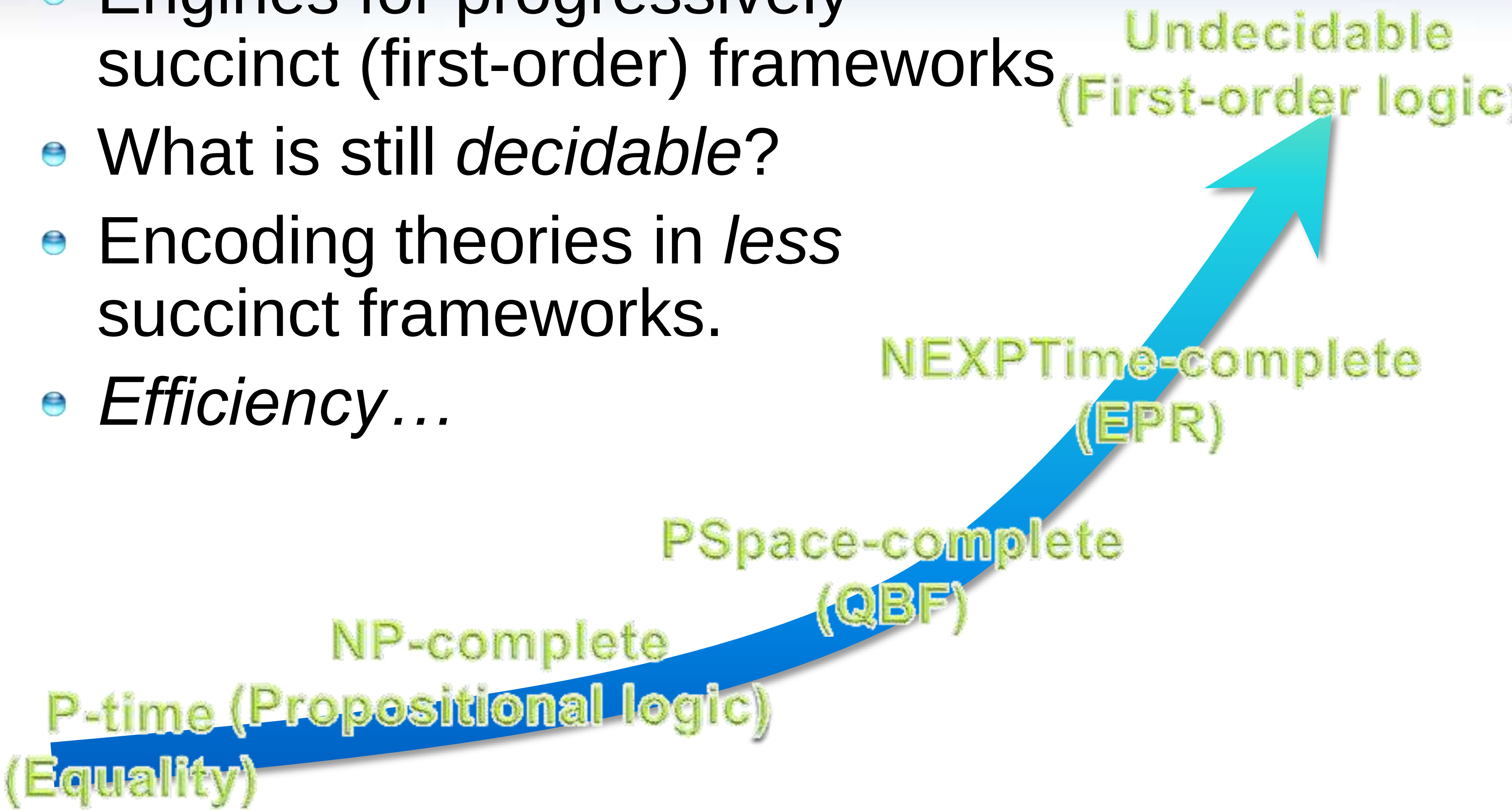
WIKIPEDIA
The Free Encyclopedia

Z3: Some Microsoft Clients

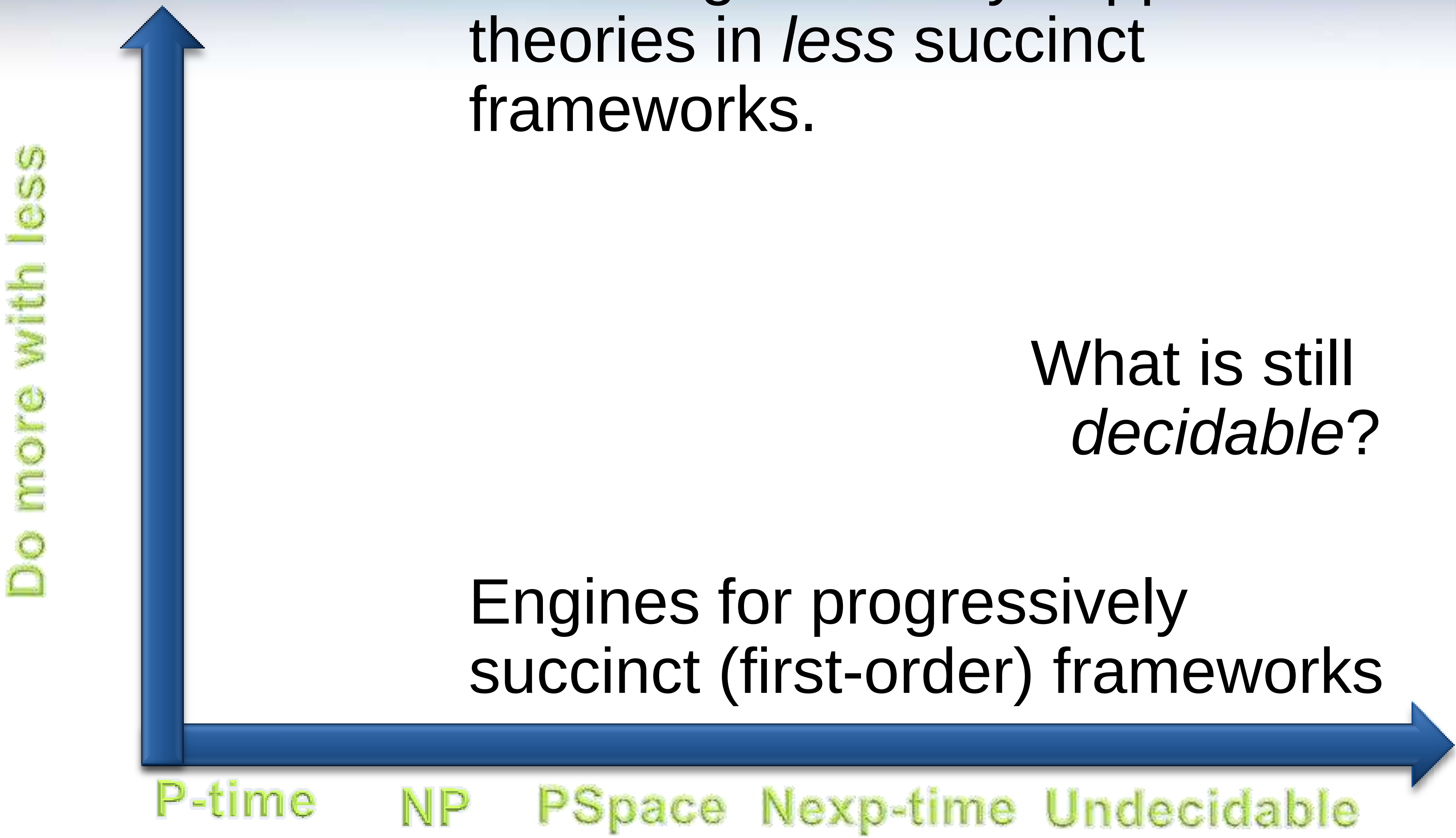


Z3 Aspirations

- Engines for progressively succinct (first-order) frameworks
- What is still *decidable*?
- Encoding theories in *less* succinct frameworks.
- *Efficiency…*



Z3/SMT Aspirations



What is SMT?

Satisfiability Modulo Theories (SMT)

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

Array Theory

Arithmetic

Uninterpreted Functions

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

$$i \neq j \Rightarrow read(write(a, i, v), j) = read(a, j)$$

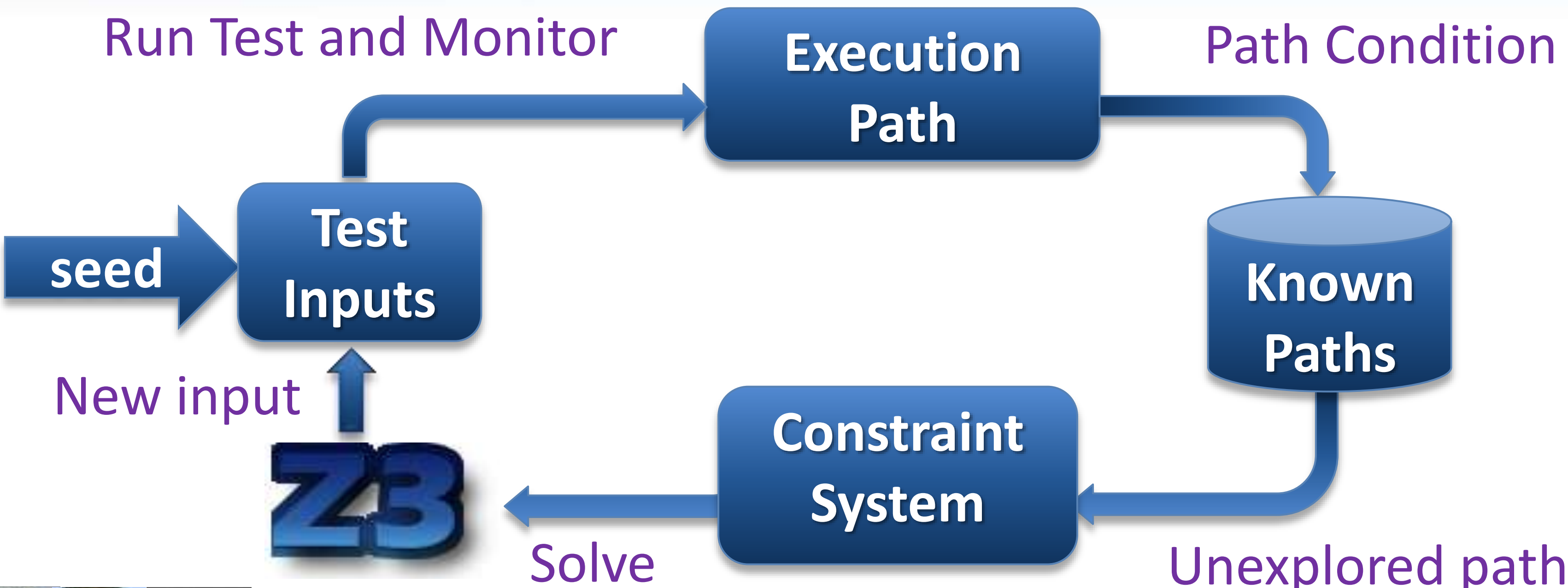
Domains from programs

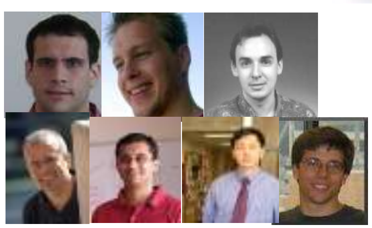
- Bits and bytes $0 = ((x-1)\ \&\ x) \Leftrightarrow x = 00100000..00$
- Numbers $x + y = y + x$
- Arrays $read(write(a,i,4),i) = 4$
- Records $mkpair(x,y) = mkpair(z,u) \Rightarrow x = z$
- Heaps $n \rightarrow^{*} n' \wedge m = cons(a,n) \Rightarrow m \rightarrow^{*} n'$
- Data-types $car(cons(x,nil))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

Application: **Dynamic Symbolic Execution**

- Pex, SAGE, Yogi, Vigilante

Dynamic Symbolic Execution





Pex Yogi Vigilante SAGE

Nikolai Tillmann Peli de Halleux (Pex), Patrice Godefroid (SAGE)
Aditya Nori, Sriram Rajamani (Yogi), Jean Philippe Martin, Miguel Castro,
Manuel Costa, Lintao Zhang (Vigilante)

Test-case generation with SAGE for exploring x86 binaries

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

Pex

SAGE

USING TEMPLATE MODELS

Challenge

Application:

Bit-precise Scalable Static Analysis

PREfix [Moy, B., Sielaff 2010]

What is wrong here?

```
int binary_se

while (low <= hig
   {
      // 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;
```

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

Package: java.util.Arrays
Function: binary_search

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

-INT_MIN=
INT_MIN

Book: Kernighan and Ritchie
Function: itoa (integer to ascii)

The PREfix Static Analysis Engine

```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)

models

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

Can Pre-condition be violated?

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

warnings

Yes: name is not initialized

Overflow on unsigned addition

m_nSize == m_nMaxSize == UINT_MAX

```
iElement = m_nSize;
if( iElement >= m_nMaxSize )
{
    bool bSuccess = GrowBuffer( iElement+1 );
    …
}
::new( m_pData+iElement ) E( element );
m_nSize++;
```

iElement + 1 == 0

Write in unallocated memory

Code was written for address space < 4GB

Using an overflowed value as 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);
```

Overflow check

Increment and exit from loop

Possible overflow

PREfix – Summary.

- 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.

ABCDE

Enabler

FAST, PRECISE SOLVER

Application

Static Program Analysis

Direction

Static Analysis Using Symbolic Execution

EFFICIENT TRUTH MAINTAINANCE

Challenge

Beneficiary

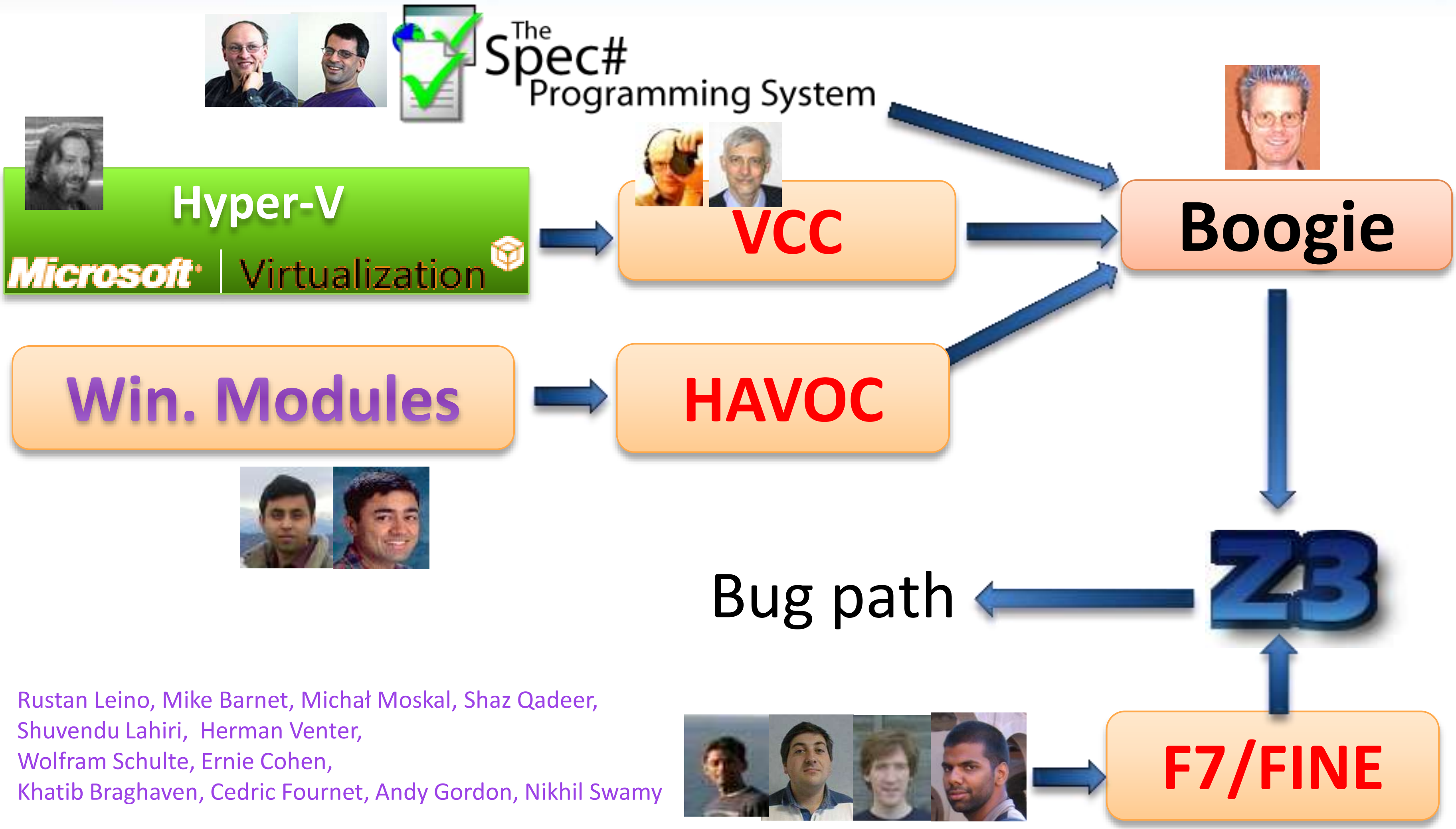
PREfix

Application:

Program Verification

- Spec#, VCC, HAVOC

Extended Static Checking and Verification



Rustan Leino, Mike Barnet, Michał Moskal, Shaz Qadeer,
Shuvendu Lahiri, Herman Venter,
Wolfram Schulte, Ernie Cohen,
Khatib Braghaven, Cedric Fournet, Andy Gordon, Nikhil Swamy

Tool Chain: Boogie

Annotated C

```
#include <vcc2.h>

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

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

VCC

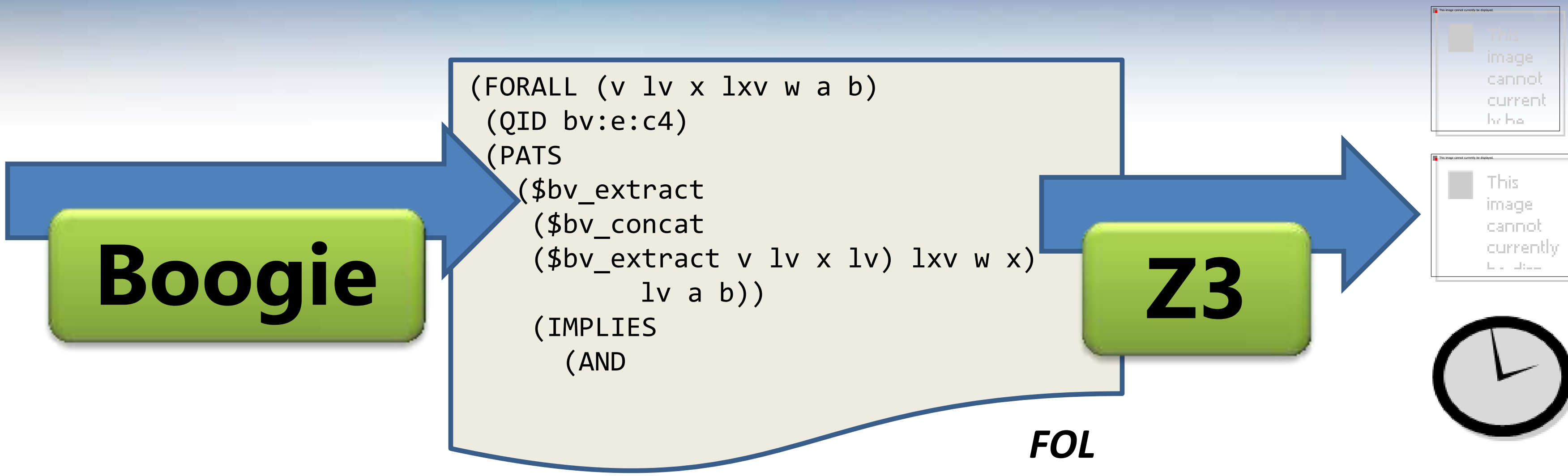
Boogie

```
$ref_cnt(old($s), #p) == $ref_cnt($s,
#p) && $ite.bool($set_in(#p,
$owns(old($s), owner)),
   $ite.bool($set_in(#p, owns),
   $st_eq(old($s), $s, #p),
   $wrapped($s, #p, $typ(#p)) &&
   $timestamp_is_now($s, #p)),
$ite.  ool($set_in(#p, owns),
      s, #p) == owner && $closed($s,
```

- Verification Condition Generator

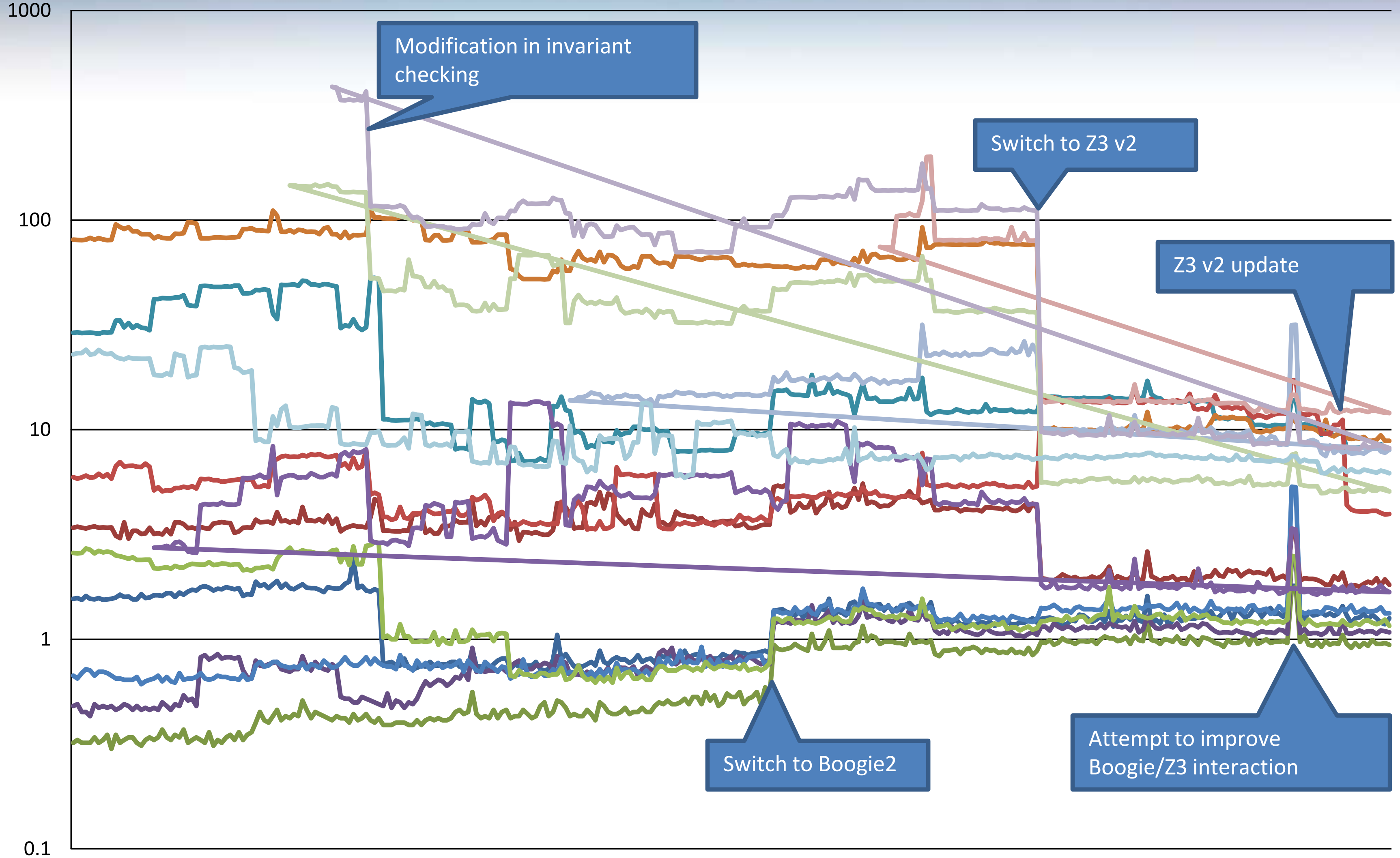
http://vcc.codeplex.com/

Tool Chain: Z3

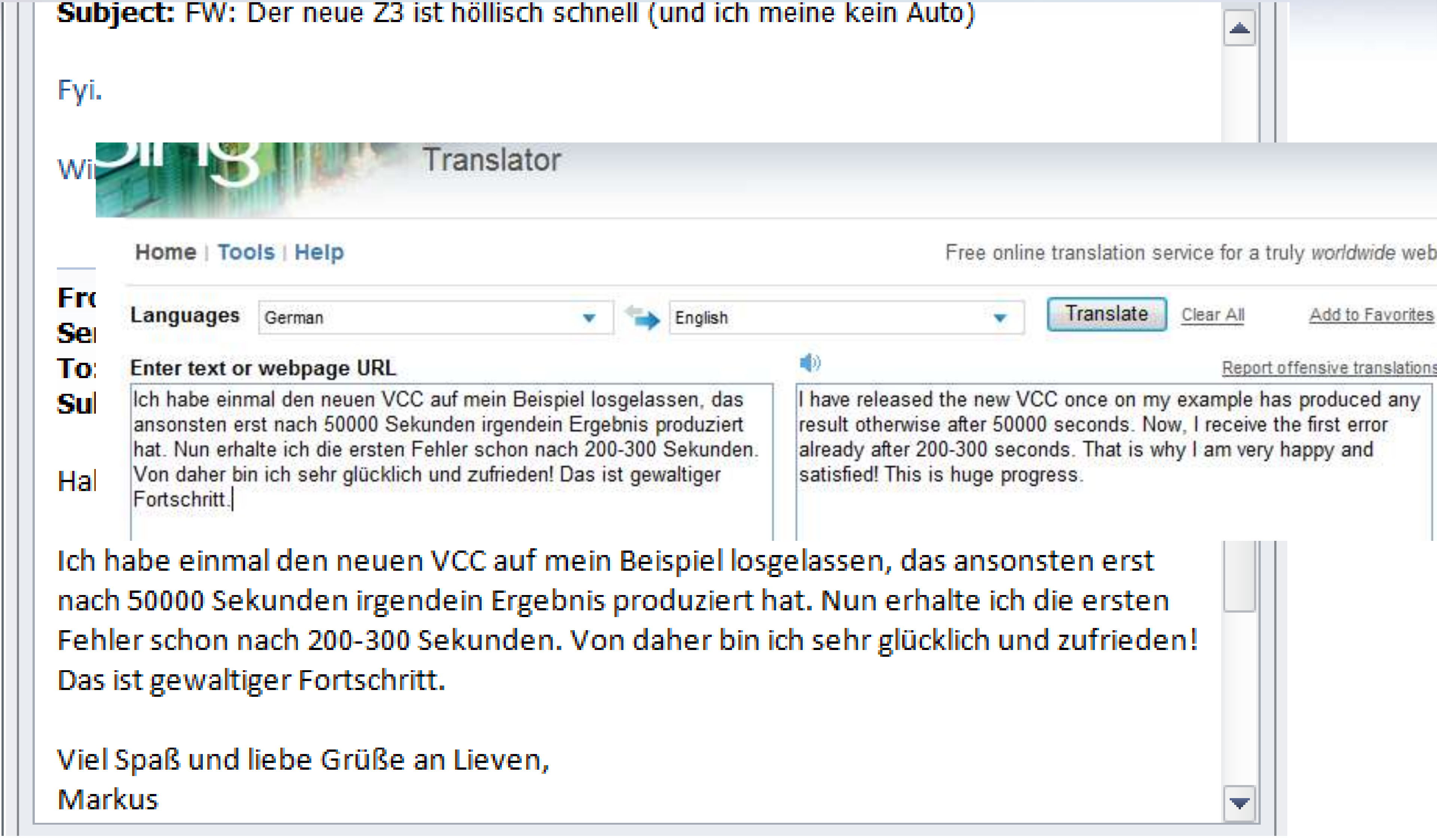


- Using Z3’s support for quantifier instantiation + theories

VCC Performance Trends Nov 08 – Mar 09



The Importance of Speed



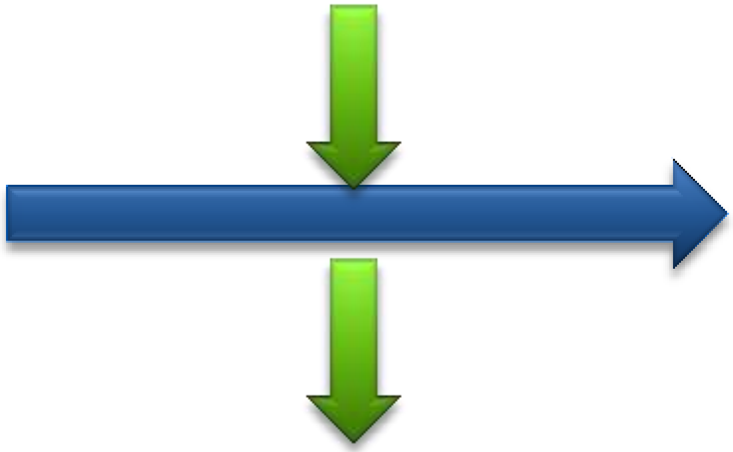
ABCDE

Enabler

QUANTIFIER
INSTANTIATION

Application

Program
Verification



Direction

Trusted OS
With
Certificates

QUANTIFIER
HEURISTICS AND COMPLETENESS

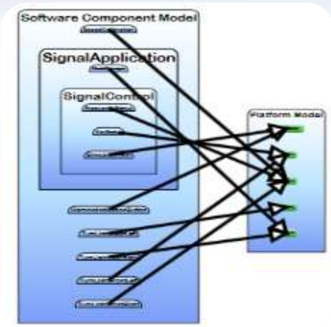
Challenge

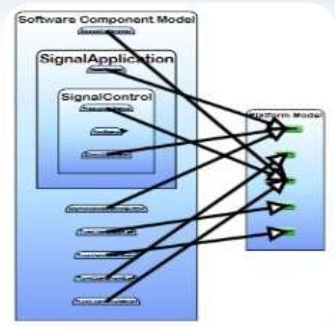
Application:

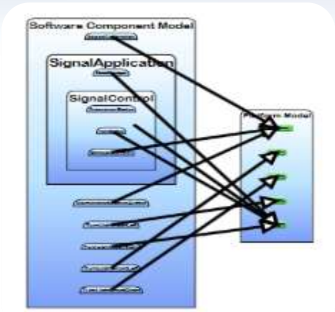
Model-Based Design

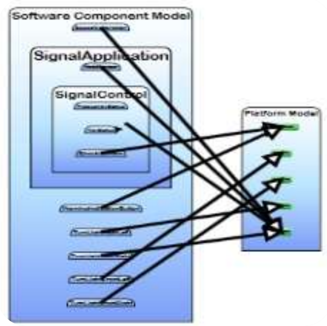
- FORMULA

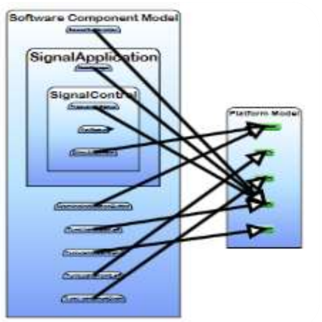
FORMULA: Design Space Exploration

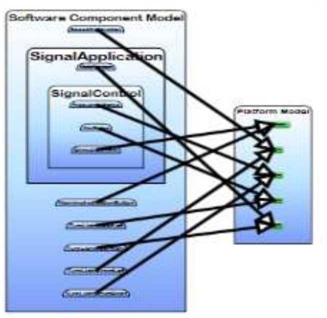






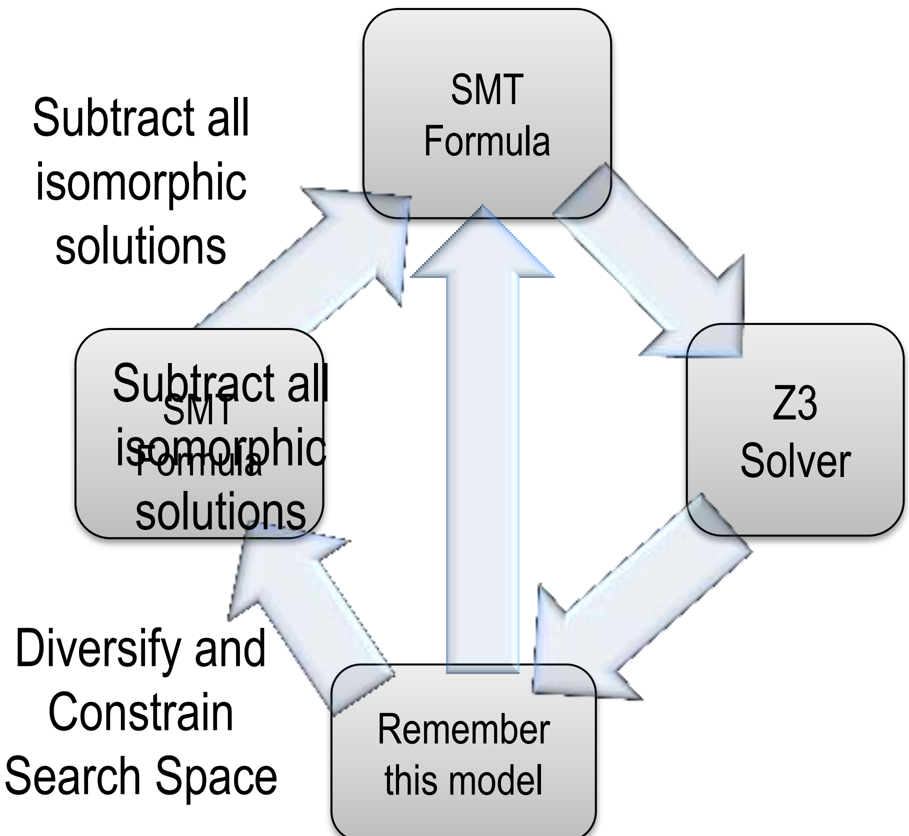


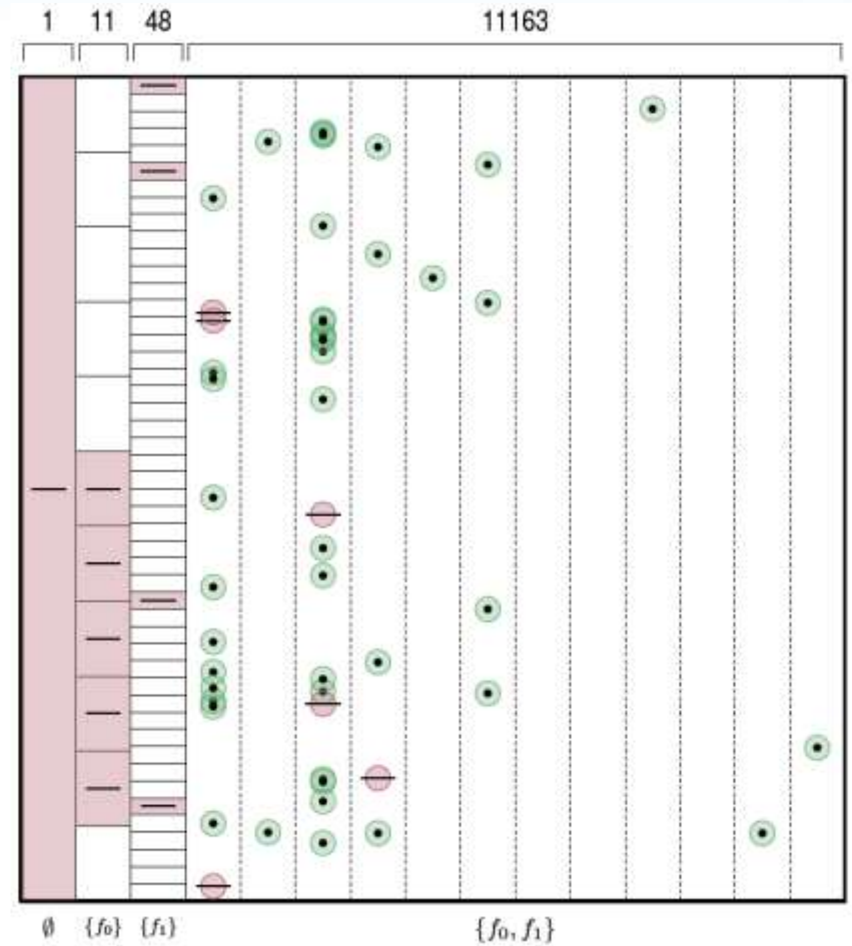




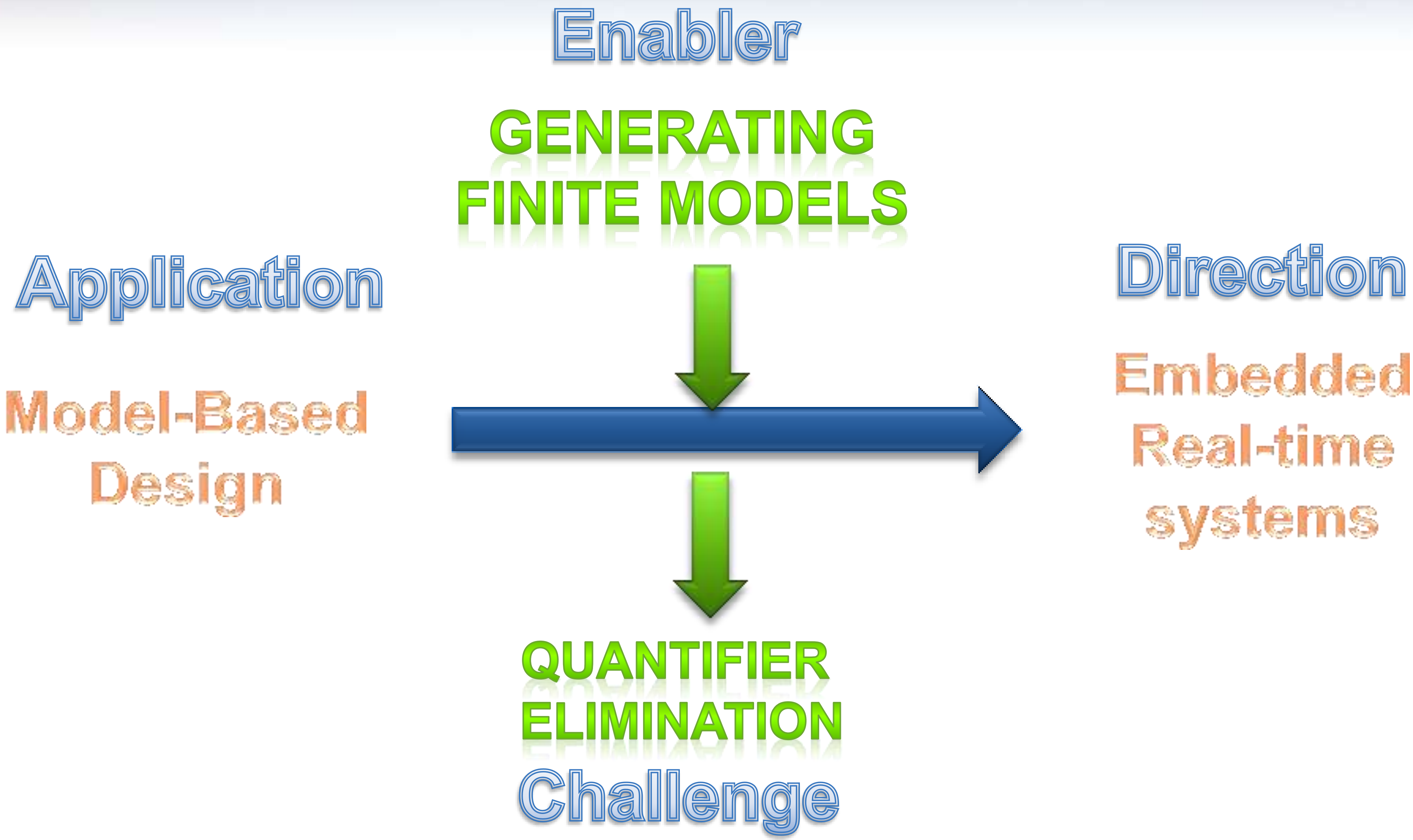
Use Design Space Exploration to identify valid candidate architectures

FORMULA: Diversified Search





ABCDE


Enabler
GENERATING FINITE MODELS
Application
Model-Based Design
Direction
Embedded Real-time systems
QUANTIFIER ELIMINATION
Challenge

Application:

Model-Based Testing

- SpecExplorer, M3

Model-based Testing and Design

Example Microsoft protocol:

- SMB2 (= remote file) Protocol Specification
- 200+ other Microsoft Protocols

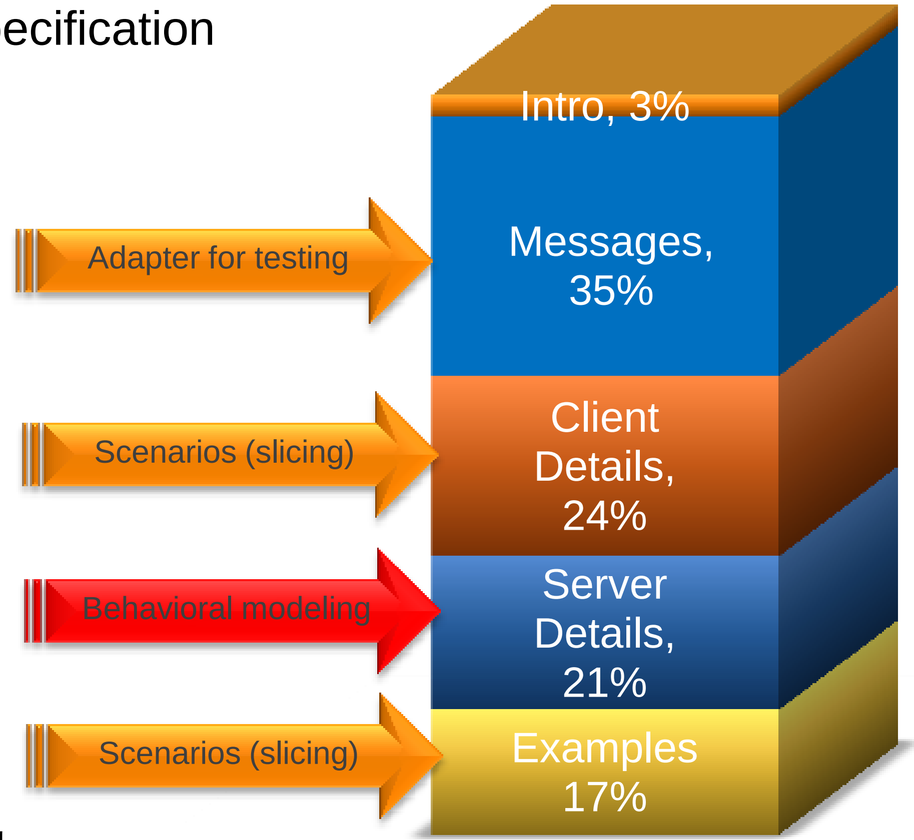
Tools:

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



Margus Veanes, Wolfgang Grieskamp

Next steps – Model-based Testing

Selected Z3 Technologies

Research around Z3

Decision Procedures

Modular Difference Logic is Hard TR 08 B, Blass Gurevich, Muthuvathi.
Linear Functional Fixed-points. CAV 09 B. & Hendrix.
A Priori Reductions to Zero for Strategy-Independent Gröbner Bases SYNASC 09 M& Passmore.
Efficient, Generalized Array Decision Procedures FMCAD 09 M & B

Combining Decision Procedures

Model-based Theory Combination SMT 07 M & B. .
Accelerating Lemma learning using DPLL(U) LPAR 08 B, Dutetre & M
Proofs, Refutations and Z3 IWIL 08 M & B
On Locally Minimal Nullstellensatz Proofs. SMT 09 M & Passmore.
A Concurrent Portfolio Approach to SMT Solving CAV 09 Wintersteiger, Hamadi & M

Quantifiers, quantifiers, quantifiers

Efficient E-matching for SMT Solvers. . CADE 07 M & B.
Relevancy Propagation. TR 07 M & B.
Deciding Effectively Propositional Logic using DPLL(Sx) IJCAR 08 M & B.
Engineering DPLL(T) + saturation. IJCAR 08 M & B.
Complete instantiation for quantified SMT formulas CAV 09 Ge & M.
On deciding satisfiability by DPLL(Γ+ T). CADE 09 Bonachina, M & Lynch.
Linear Quantifier Elimination as Abstract Decision Proc. IJCAR 10, B. .

.

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

Combinatory Array Logic

- A basis of operations

$$write(a,i,v) = \lambda j.ite(i = j, v, a[j])$$

$$K(v) = \lambda j.v$$

$$map_f(a,b) = \lambda j.f(a[j], b[j])$$

$$\delta(a) = a[\varepsilon(a)]$$

[FMCAD 2009]

Combinatory Array Logic

- Derived operations

$\emptyset$	$\triangleq$	$K(\mathit{false})$	$\emptyset_{Bag}$	$\triangleq$	$K(0)$
$\{a\}$	$\triangleq$	$\mathit{write}(\emptyset, a, \mathit{true})$	$\{a\}$	$\triangleq$	$\mathit{write}(\emptyset, a, 1)$
$a \in A$	$\triangleq$	$A[a]$	$\mathit{mult}(a, A)$	$\triangleq$	$A[a]$
$A \cup B$	$\triangleq$	$\mathit{map}_{\vee}(A, B)$	$A \oplus B$	$\triangleq$	$\mathit{map}_{+}(A, B)$
$A \cap B$	$\triangleq$	$\mathit{map}_{\wedge}(A, B)$	$A \sqcap B$	$\triangleq$	$\mathit{map}_{\min}(A, B)$
$\mathit{finite}(A)$	$\triangleq$	$(\delta(A) = \mathit{false})$	$\mathit{finite}_{Bag}(A)$	$\triangleq$	$(\delta(A) = 0)$

Efficient E-graph Matching

- Match: ***read(write(A,I,V),I) = read(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: ***read(write(A,I,V),I) = 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: ***read(write(A,I,V),I) = 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 sub-terms upwards to roots in index.

[CADE 2007]

Efficient E-graph Matching

- Match: ***read(write(A,I,V),I) = read(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 sub-terms upwards to roots in index.

[CADE 2007]

Efficient E-graph Matching

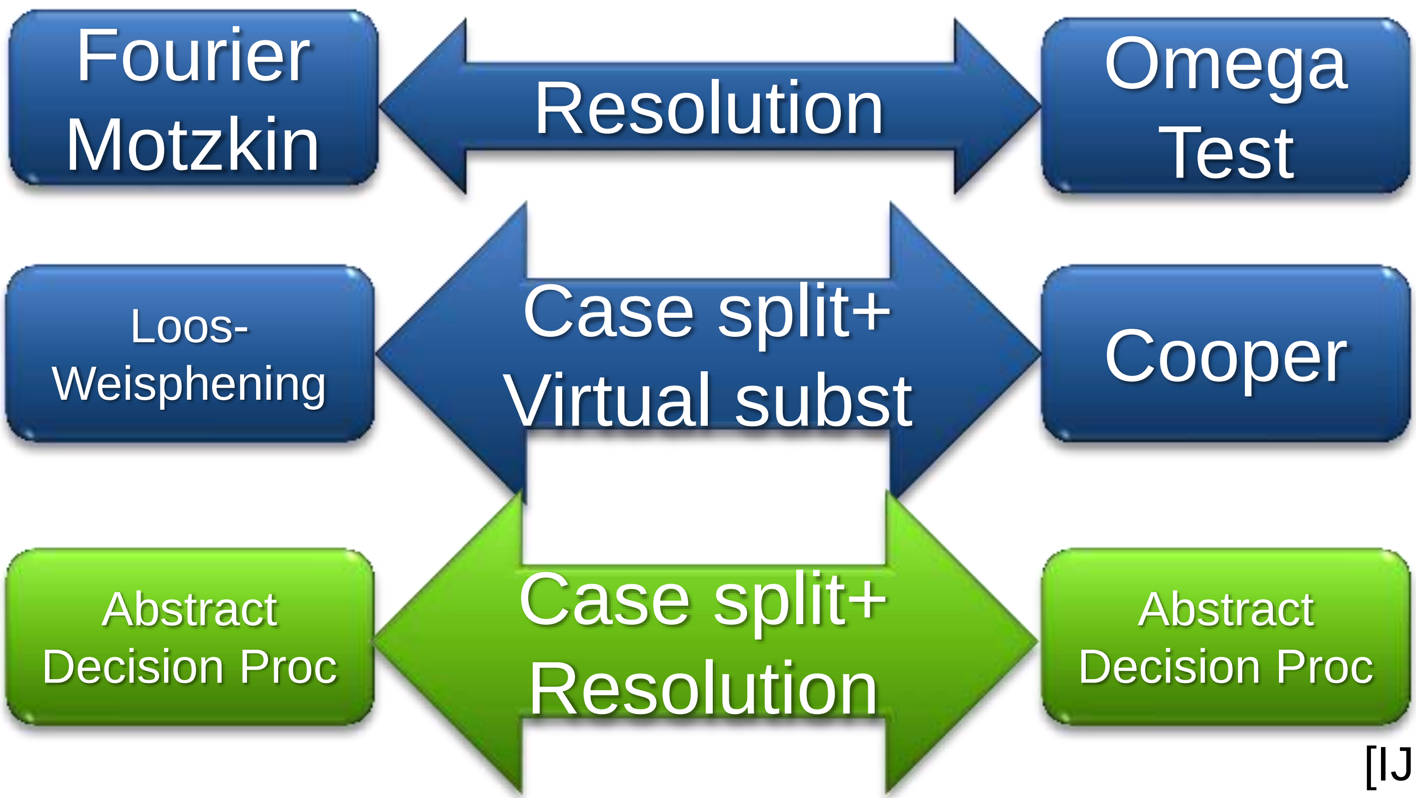
- Match: ***read(write(A,I,V),I) = read(write(a,g(c),c),g(c))***
  Assuming
  - $E = \{ g(a) = f(b, c), b = d, a = c \}$

- 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]

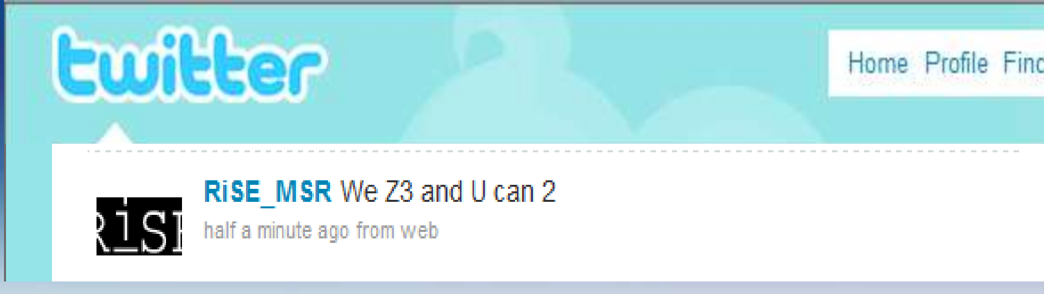
Linear quantifier Elimination as an Abstract Decision Procedure

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



[IJCAR 2010]

Conclusions



- 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