Experiments in Software Verification using SMT Solvers
VS Experiments 2008 – Toronto, Canada

Leonardo de Moura
Microsoft Research
What is SMT?

Experiments:
- Windows kernel verification.
  - Extending SMT solvers.
- Garbage collector (Singularity) verification
  - Supporting decidable fragments.
Satisfiability Modulo Theories (SMT)

SAT + Theories = SMT

- Arithmetic
- Bit-vectors
- Arrays
- ...

Experiments in Software Verification using SMT Solvers
Satisfiability Modulo Theories (SMT)

\[ x + 2 = y \implies f(\text{read}(\text{write}(a, x, 3), y - 2)) = f(y - x + 1) \]
\[ x + 2 = y \Rightarrow f(read(write(a, x, 3), y - 2)) = f(y - x + 1) \]
\( x + 2 = y \Rightarrow f(read(write(a, x, 3), y - 2)) = f(y - x + 1) \)
Z3 is a new solver developed at Microsoft Research.
Development/Research driven by internal customers.
Free for academic research.
Interfaces:

http://research.microsoft.com/projects/z3
HAVOC
Verifying Windows Components

Lahiri & Qadeer, POPL’08,
Also: Ball, Hackett, Lahiri, Qadeer, MSR-TR-08-82.
HAVOC's Architecture

Experiments in Software Verification using SMT Solvers
typedef struct _LIST_ENTRY{
    struct _LIST_ENTRY *Flink, *Blink;
} LIST_ENTRY, *PLIST_ENTRY;

typedef struct _NODEA{
    PRESOURCE Resource;
    LIST_ENTRY NodeBQueue;
    ...
} NODEA, *PNODEA;

typedef struct _NODEB{
    PNODEA ParentA;
    ULONG State;
    LIST_ENTRY NodeALinks;
    ...
} NODEB, *PNODEB;

#define CONTAINING_RECORD(addr, type, field)\    \((type *)((PCHAR)(addr) - (PCHAR)(&((type *)0)->field)))

Doubly linked lists in Windows Kernel code
 Pointer Arithmetic
q = CONTAINING_RECORD(p, IRP, link)
   = (IRP *) ((char*)p – (char*)(&(IRP *)0)→link))

 Transitive Closure
Reach(next, u) ≡ {u, u→next, u→next→next, ...}
forall (x, Reach(next,p), CONTAINING_RECORD(x, IRP, link)->state == PENDING)
Annotation Language & Logic

- Procedure contracts
  - requires, ensures, modifies
- Arbitrary C expressions
  - program variables, resources
  - Boolean connectives
  - quantifiers
- Can express a rich set of contracts
  - API usage (e.g. lock acquire/release)
  - Synchronization protocols
  - Memory safety
  - Data structure invariants (linked list)
- Challenge:
  - Retain efficiency
  - Decidable fragments
Efficient logic for program verification

- Logic with Reach, Quantifiers, Arithmetic
  - Expressive
  - Careful use of quantifiers

Encoding using quantifiers and triggers
Success Story

- Used to check Windows Kernel code.
- Found 50 bugs, most confirmed.
  - 250 lines required to specify properties.
  - 600 lines of manual annotations.
  - 3000 lines of inferred annotations.

Houdini-like algorithm (Flanagan, Leino)
Extending Z3

- Axioms
- Inference rules (not supported yet)
- Very lazy loop
- New Z3 theory (too complicated for users)
Axioms

- Easy if theory can be encoded in first-order logic.
- Example: partial orders.
  \[ \forall x: p(x,x) \]
  \[ \forall x,y,z: p(x,y), p(y,z) \Rightarrow p(x,z) \]
  \[ \forall x,y: p(x,y), p(y,x) \Rightarrow x = y \]

Problems:
- Is E-matching or SP a decision procedure for this theory?
- Model extraction
- Efficiency
Some users (e.g., HAVOC) want to provide inference rules to Z3.

More flexibility (e.g., side conditions)

High level language for implementing custom decision procedures.
Adding a theory $T$:

1. Replace $T$ symbols with uninterpreted symbols.
2. Invoke Z3.
3. If unsatisfiable, then return UNSAT.
4. Inspect the model + implied equalities (i.e., assigned literals and equalities).
5. Check if the assigned theory literals + equalities are satisfiable.
6. If they are, then return SAT.
7. Otherwise, add a new lemma and/or implied equalities go back to step 2.

Model Based Theory Combination [SMT’08]
Very lazy loop (example)

\[ \neg p(a,f(c)) \lor \neg p(a, f(d)) \]
\[ p(a,b) \]
\[ p(b,f(c)) \]
\[ c = d \]

\[ \neg p(a,f(c)) \lor \neg p(a, f(d)) \]
\[ p(a,b) \]
\[ p(b,f(c)) \]
\[ c = d \]
\[ \neg p(a,b) \lor \neg p(b,f(c)) \lor p(a,f(c)) \]

Model:
\[ \neg p(a,f(c)), p(a,b), p(b,f(c)), b = c, f(c) = f(d) \]

T-Lemma:
\[ \neg p(a,b) \lor \neg p(b,f(c)) \lor p(a,f(c)) \]

\[ \neg p(a,b) \lor \neg p(b,f(c)) \lor p(a,f(c)) \]

z3
unsat

Experiments in Software Verification using SMT Solvers
Verifying Garbage Collectors
- Automatically and fast

Singularity

- Safe micro-kernel
  - 95% written in C#
  - all services and drivers in processes

- Software isolated processes (SIPs)
  - all user code is verifiably safe
  - some unsafe code in trusted runtime
  - processes and kernel sealed at execution
  - static verification replaces hardware protection
  - all SIPs run in ring 0

- Communication via channels
  - channel behavior is specified and checked
  - fast and efficient communication

- Working research prototype
  - not Windows replacement
  - shared source download
Context

Bartok
- MSIL → X86 Compiler

BoogiePL
- Procedural low-level language
- Contracts
- Verification condition generator

Garbage Collectors
- Mark&Sweep
- Copying GC
- Verify small garbage collectors
  - more automated than interactive provers
  - borrow ideas from type systems for regions
Goal: safely run untrusted code

untrusted code

typed x86

untrusted code

trusted computing base (minimize this!)

MSIL

compiler

I/O

exception handling

garbage collector

linker, loader

safety verifier

MSIL: MSFT Intermediary Language
Mark-sweep and copying collectors

abstract graph (root)

mark-sweep

copying from
copying to
Garbage collector properties

- safety: gc does no harm
  - type safety
    - gc turns well-typed heap into well-typed heap
  - graph isomorphism
    - concrete graph represents abstract graph

- effectiveness
  - after gc, unreachable objects reclaimed

- termination

- efficiency

verified

not verified
Abstract graph

Proving safety

Concrete graph

procedure GarbageCollectMs()
  requires MsMutatorInv(root, Color, $toAbs, $AbsMem, Mem);
  modifies Mem, Color, $toAbs;
  ensures ...;
  { ...
    call Mark(root);
    call Sweep();
  }

function MsMutatorInv(...) returns (bool) {
  WellFormed($toAbs) && memAddr(root) && $toAbs[root] != NO_ABS
  && (forall i:int::{memAddr(i)} memAddr(i) ==> ObjInv(i, $toAbs, $AbsMem, Mem))
  && (forall i:int::{memAddr(i)} memAddr(i) ==> White(Color[i]))
  && (forall i:int::{memAddr(i)} memAddr(i) ==> ($toAbs[i]==NO_ABS <=>
    Unalloc(Color[i])))
}

function ObjInv(...) returns (bool) {
  memAddr(i) && $toAbs[i] != NO_ABS ==> ...
  ... $toAbs[Mem[i, field1]] != NO_ABS ... 
  ... $toAbs[Mem[i, field1]] == $AbsMem[$toAbs[i, field1]] ... 
}
Idea: use marker

function T(i:int) returns (bool) { true }

Relativize quantifiers using marker

function GcInv(Color:[int]int, $toAbs:[int]int, $AbsMem:[int,int]int, Mem:[int,int]int) returns (bool) {
    WellFormed($toAbs)
    && (forall i:int::{T(i)} T(i) ==> memAddr(i) ==> ObjInv(i, $toAbs, $AbsMem, Mem)
        && 0 <= Color[i] && Color[i] < 4
        && (Black(Color[i]) ==> !White(Color[Mem[i,0]]) && !White(Color[Mem[i,1]])
        && ($toAbs[i] == NO_ABS <=> Unalloc(Color[i]))
    )
}
procedure Mark(ptr:int)
    requires GcInv(Color, $toAbs, $AbsMem, Mem);
    requires memAddr(ptr) && T(ptr);
    requires $toAbs[ptr] != NO_ABS;
    modifies Color;
    ensures GcInv(Color, $toAbs, $AbsMem, Mem);
    ensures (forall i:int::{T(i)} T(i) ==> !Black(Color[i]) ==> Color[i] == old(Color)[i]);
    ensures !White(Color[ptr]);
{
    if (White(Color[ptr])) {
        Color[ptr] := 2; // make gray
        call Mark(Mem[ptr,0]);
        call Mark(Mem[ptr,1]);
        Color[ptr] := 3; // make black
    }
}
EPR (Effectively Propositional)
  Aka: Bernays–Schönfinkel class
Stratified EPR
Array Property Fragment
Stratified Array Property Fragment

It can be used to verify the GC properties!
Prefix $\exists^* \forall^*$ + no function symbols.

Examples:
- $\forall x, y, z: \neg p(x, y)$ or $\neg p(y, z)$ or $p(x, z)$
- $\forall x: \neg p(x, a)$ or $\neg q(x, b)$

Why is it useful?
- Model checking problems
- QBF
- Finite model finding
- Useful theories: partial orders.
Finite Herbrand Universe.

\[ \forall x: \neg p(x,a) \text{ or } \neg q(x,b) \]
\[ \forall x: p(x,x) \]
\[ p(c,a) \text{ or } q(c,b) \]

Herbrand Universe \{a, b, c\}

\[ \neg p(a,a) \text{ or } \neg q(a,b) \]
\[ \neg p(b,a) \text{ or } \neg q(b,b) \]
\[ \neg p(c,a) \text{ or } \neg q(c,b) \]
\[ p(a,a) \]
\[ p(b,b) \]
\[ p(c,c) \]
\[ p(c,a) \text{ or } q(c,b) \]

Exponential blowup

SAT-solver

Experiments in Software Verification using SMT Solvers
EPR: efficient implementation

- DPLL(SX) calculus: DPLL + substitution sets (BDDs) [IJCAR’08]

\[ \forall x: \neg p(x, a) \lor \neg q(x, b) \]

\[ \forall x: \neg p(x1, x2) \lor \neg q(x3, x4) + \{(a, a, a, b), (b, a, b, b), (c, a, c, b)\} \]

Substitution set (BDD)
Many sorted first order logic.

$S_1 < S_2$ if there is a function $f : \ldots S_1 \ldots \rightarrow S_2$

A formula is **stratified** if there is no sort $S$ s.t. $S < S$

A stratified formula has a finite Herbrand Universe.

**Example:**

\[
\forall x \; S_1 : f(g(x)) = a \\
g(b) = c \\
\text{where:} \\
g : S_1 \rightarrow S_2 \\
f : S_2 \rightarrow S_3 \\
a : S_3 \\
b : S_1 \\
c : S_2
\]

Herbrand Universe:

\{ a, b, c, g(b), f(g(b)), f(c) \}
Sort inference + restrictions

Problematic example:

\[ \forall x, y: f(x) \neq f(y) \lor x = y \]
\[ \forall x: f(x) \neq c \]
\[ \forall x: x = a \]

\[ \forall x S_1, y S_1: f(x) \neq f(y) \lor x = y \]
\[ \forall x S_1: f(x) \neq c \]
\[ \forall x S_3: x = a \]
\[ f : S_1 \to S_2 \]
\[ c : S_2 \]
\[ a : S_3 \]

Cardinality Constraint
Almost there...

\[
\forall i: \text{int} \cdot T(i) \implies \text{memAddr}(i) \implies
\begin{align*}
\text{ObjInv}(i, \$\text{toAbs}, \$\text{AbsMem}, \text{Mem}) \\
\&\& 0 \leq \text{Color}[i] \&\& \text{Color}[i] < 4 \\
\&\& (\text{Black}(\text{Color}[i]) \implies !\text{White}(\text{Color}[\text{Mem}[i,0]]) \&\& !\text{White}(\text{Color}[\text{Mem}[i,1]]) \\
\&\& ($\text{toAbs}[i] == \text{NO_ABS} \iff \text{Unalloc}(\text{Color}[i]))
\end{align*}
\]

\[
\forall i: \text{Addr} \\
\begin{align*}
\text{ObjInv}(i, \$\text{toAbs}, \$\text{AbsMem}, \text{Mem}) \\
\&\& (\text{color}[i] = \text{black} \text{ or } \text{color}[i] = \text{white} \text{ or } \text{color}[i] = \text{gray}) \\
\&\& (\text{Black}(\text{color}[i]) \implies !\text{White}(\text{color}[\text{Mem}[i,f0]]) \&\& !\text{White}(\text{Color}[\text{Mem}[i,f1]]) \\
\&\& ($\text{toAbs}[i] == \text{NO_ABS} \iff \text{Unalloc}(\text{Color}[i]))
\end{align*}
\]
Array Property Fragment (APF)

\[ \forall i_1, \ldots, i_n : F[i_1, \ldots, i_n], \]

*F* is in NNF, then the following atoms can contain universal variables:

- \( i_k > t \) (t is ground)
- \( i_k > i_k' \)
- \( i_k \neq t \) (t is ground)
- \( i_k \neq i_k' \)
- \( L[a[i_k]] \) (\( i_k \) only appears in \( a[i_k] \))
Array is sorted:

- $\forall i, j: i \leq j$ implies $a[i] \leq a[j]$, or equivalently:
  - $\forall i, j: i > j$ or $a[i] \leq a[j]$

Array update $b = \text{write}(a, j, v)$

- $b[j] = v$
- $\forall x: x > j-1$ or $b[x] = a[x]$
- $\forall x: x < j+1$ or $b[x] = a[x]$

Equivalent to:

- $\forall x: x = j$ or $b[x] = a[x]$
Stratified APF

- Yeting Ge (Intern 2008)
- Nested (stratified) arrays in APF.
- Stratified EPR + some arithmetic.
- Example:
  \[ \forall i, j: i \leq j \implies a[a'[i]] \leq a[a'[j]] \]
- It supports other extensions for pointer arithmetic.
Users frequently need new theories.
- Quantifiers.
- Inference rules.
- Very lazy loop.

Decidable fragments are useful in practice.

http://research.microsoft.com/projects/z3

Thank You!
Is Z3 available for commercial use?

Not yet...

However,

PEX (comes with Z3) and Chess will be available for commercial use for VS users.

- [http://research.microsoft.com/Pex/](http://research.microsoft.com/Pex/)

SLAM/SDV 2.0 (comes with Z3) is part of DDK and will ship with the next version of Windows.

- [http://research.microsoft.com/slam/](http://research.microsoft.com/slam/)