Engineering Theories with Z3

Lecture 6

Nikolaj Bjørner
Microsoft Research
DTU Winter course January 10th 2012
Organized by Flemming Nielson & Hanne Riis Nielson
Plan

- Write a custom theory solver with propagation.
- Encode theories using base theories
- Write a custom theory solver without propagation
Eager vs. Lazy reduction approach to theory solving:

- Eager reduction: encode theory using existing theories
- Lazy reduction: lazily unfold theory using special purpose procedures
Many techniques apply broadly to SMT solvers: Barcelogic, CVC, Ergo, Mathsat, OpenSMT, Yices, ..

Many tools already use techniques ..

.. But many more tools should really do it too.
Why Engineering Theories?

Support

**Rich Theories** *(and logics)* with **Efficient** Decision Procedures

- Auth
- MSOL
- Sequences
- XDucers
- Queues
- ASP
- DL
- homomorphisms
- Optimization
- Orders
- Objects
- HOL
- MultiSets
- BAPA
- Strings
- Reg. Exprs.
- NRA
- NIA
- Floats
- f*
- *
- SAT
- EUF
- LRA
- LIA
- Arrays
- Bit-Vectors
- Alg. DT
We review three methods:

Theory Solver: Optimization, Partial Orders

Reduction: Object Types

Saturation: HOL
The MUNCH Tool: automated reasoner for collections

This is the web page for the MUNCH tool. Currently the following is available for download:

- paper describing the tool
- implementation
- some examples and their output

Examples are written in the separate file (examples.txt). The tool then parses this input into a language corresponding to the grammar described in the paper and in the file ASTMultisets.scala. MUNCH invokes z3.

Playing with the MUNCH tool

The MUNCH tool is written in Scala and for testing MUNCH you need to have Scala installed. To run MUNCH, on your machine, first download the source code and compile it.
Overview of methods

New Theory

Theory Solver (1st class solver)

Reduction (eager reduction)

Saturation (lazy reduction)

Z3

Compile

Constraints

Equalities

Search

Model

Partial Compile

New Theory

New Theory

New Theory
Optimization

Get More Satisfaction with SMT

Intro
SMT?
Z3?

Lazy Reduction
Eager Reduction
Theory Solver

SMT?

Oliveras, Nieuwenhuis, SAT 2006
<table>
<thead>
<tr>
<th>Name</th>
<th>Formula</th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_0$</td>
<td>$a \lor b \lor x \geq 2$</td>
<td>$\infty$</td>
</tr>
<tr>
<td>$F_1$</td>
<td>$\neg a \lor x \geq 3$</td>
<td>3</td>
</tr>
<tr>
<td>$F_2$</td>
<td>$\neg b \lor x \geq 3$</td>
<td>4</td>
</tr>
<tr>
<td>$F_3$</td>
<td>$x &lt; 2$</td>
<td>5</td>
</tr>
</tbody>
</table>

Unsat
Table: Weighted MaxSMT Formulas

<table>
<thead>
<tr>
<th>Name</th>
<th>Formula</th>
<th>weight</th>
<th>Penalty</th>
<th>Satisfiability</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_0$</td>
<td>$a \lor b \lor x \geq 2$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\neg a \land \neg b \land x &lt; 2$</td>
</tr>
<tr>
<td>$F_1$</td>
<td>$\neg a \lor x \geq 3$</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$F_2$</td>
<td>$\neg b \lor x \geq 3$</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$F_3$</td>
<td>$x &lt; 2$</td>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Weighted MaxSMT

<table>
<thead>
<tr>
<th>Name</th>
<th>Formula</th>
<th>weight</th>
<th>Objective</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_0$</td>
<td>$a \lor b \lor x \geq 2$</td>
<td>$\infty$</td>
<td>$\text{Sat } \neg a \land b \land x = 2$</td>
</tr>
<tr>
<td>$F_1$</td>
<td>$\neg a \lor x \geq 3$</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>$F_2$</td>
<td>$\neg b \lor x \geq 3$</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>$F_3$</td>
<td>$x &lt; 2$</td>
<td>5</td>
<td>$\text{Penalty: } 9 = 4 + 5$</td>
</tr>
</tbody>
</table>
## Weighted MaxSMT

<table>
<thead>
<tr>
<th>Name</th>
<th>Formula</th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_0$</td>
<td>$a \lor b \lor x \geq 2$</td>
<td>$\infty$</td>
</tr>
<tr>
<td>$F_1$</td>
<td>$\neg a \lor x \geq 3$</td>
<td>3</td>
</tr>
<tr>
<td>$F_2$</td>
<td>$\neg b \lor x \geq 3$</td>
<td>4</td>
</tr>
<tr>
<td>$F_3$</td>
<td>$x &lt; 2$</td>
<td>5</td>
</tr>
</tbody>
</table>

Sat $\neg a \land \neg b \land x \geq 2$

Penalty: 5
<table>
<thead>
<tr>
<th>Name</th>
<th>Formula</th>
<th>weight</th>
<th>Penalty: 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_0$</td>
<td>$a \lor b \lor x \geq 2$</td>
<td>$\infty$</td>
<td></td>
</tr>
<tr>
<td>$F_1$</td>
<td>$\neg a \lor x \geq 3$</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>$F_2$</td>
<td>$\neg b \lor x \geq 3$</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>$F_3$</td>
<td>$x &lt; 2$</td>
<td>5</td>
<td>$\text{Sat } a \land \neg b \land x &lt; 2$</td>
</tr>
</tbody>
</table>
Weighted MaxSMT

<table>
<thead>
<tr>
<th>Formula</th>
<th>weight</th>
<th>Initially: All atoms are unassigned</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a \lor b \lor x \geq 2$</td>
<td>$\infty$</td>
<td>$Cost = 0$</td>
</tr>
<tr>
<td>$F_1 \lor \neg a \lor x \geq 3$</td>
<td>$3$</td>
<td>$Assert \neg a \land b \land x &lt; 2$</td>
</tr>
<tr>
<td>$F_2 \lor \neg b \lor x \geq 3$</td>
<td>$4$</td>
<td>$Propagate: F_2: Cost := Cost + 4 := 4$</td>
</tr>
<tr>
<td>$F_3 \lor x &lt; 2$</td>
<td>$5$</td>
<td>$Best \ so \ far: MinCost = 4$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$Add \ Axiom \neg F_2 \ - \ backtrack$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$Assert \ F_3 \ Cost = 5 &gt; MinCost$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$Add \ Axiom \neg F_3 \ - \ backtrack$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$.... \ Assert a \land \neg b \land x &lt; 2 \land F_1$</td>
</tr>
</tbody>
</table>

What does it take to encode this in Z3?

```plaintext
let block() =
    let offender = optimize_cost ctx costs min_cost
    th.AssertTheoryAxiom(ctx.MkNot(ctx.MkAnd offender))
let Assign p vl =
    if vl then
        let = th.NewAssignment <- (fun p vl -> Assign p vl)
        let _ =
            if cost > min_cost then
                block()
            cost <- cost + w;
            tail.Add (fun () -> cost <- cost - w)
            tail.Add (fun () -> costs <- List.tail costs)
    let _ =
        if cost > min_cost then
            block()
```
Principles of Modern SMT solvers in two slides
Modern DPLL in a nutshell

Initialize \( \epsilon | F \)  

Decide \( M | F \rightarrow M, \ell | F \)  

Propagate \( M | F, C \vee \ell \rightarrow M, \ell^{C\vee\ell} | F, C \vee \ell \)  

Conflict \( M | F, C \rightarrow M | F, C | C \)  

Resolve \( M | F | C' \vee \neg \ell \rightarrow M | F | C' \vee C \)  

Learn \( M | F | C \rightarrow M | F, C | C \)  

Backjump \( M \neg \ell M' | F | C \vee \ell \rightarrow M \ell^{C\vee\ell} | F \)  

Unsat \( M | F | \emptyset \rightarrow Unsat \)  

Sat \( M | F \rightarrow M \)  

Restart \( M | F \rightarrow \epsilon | F \)

F is a set of clauses
\( \ell \) is unassigned
C is false under M
C is false under M
\( \ell^{C\vee\ell} \in M \)

Adapted and modified from [Nieuwenhuis, Oliveras, Tinelli J.ACM 06]
**DPLL(\(T\)) solver interaction**

**T- Propagate**

\[
M \mid F, C \vee \ell \Rightarrow M, \ell^{C \vee \ell} \mid F, C \vee \ell \quad \text{\(C\) is false under \(T + M\)}
\]

**T- Conflict**

\[
M \mid F \Rightarrow M \mid F \mid \neg M' \quad \text{\(M' \subseteq M\) and \(M'\) is false under \(T\)}
\]

**T- Propagate**

\[
a > b, b > c \mid F, a \leq c \vee b \leq d \Rightarrow
\]

\[
a > b, b > c, b \leq d^{a \leq c \vee b \leq d} \mid F, a \leq c \vee b \leq d
\]

**T- Conflict**

\[
M \mid F \Rightarrow M \mid F, a \leq b \vee b \leq c \vee c < a
\]

\[
\text{where} \quad a > b, b > c, a \leq c \subseteq M
\]
How does Z3 enable $T$ solvers?
DPLL(T) Solver Interaction

Calls into DPLL engine

Callbacks from DPLL engine with new assignment

T-Propagate

T-Propagate

T-Conflict

T-Conflict

Calls into DPLL engine

Callbacks from DPLL engine

``` Ocaml
let mutable cost = 0
let mutable costs = []
let mutable min_cost = Int32.max_int
let mutable weights = Dictionary<Term,int>()
let th = ctx.MkTheory("opt")
let block() =
  let offender = optimize_cost ctx costs min_cost
  th.AssertTheoryAxiom(ctx.MkNot(ctx.MkAnd offender))
let Assign p vl =
  if vl then
    let w = weights.[p]
    cost <- cost + w;
    trail.Add (fun () -> cost <- cost - w)
    costs <- (w,p)::costs;
    trail.Add (fun () -> costs <- List.tail costs)
    if cost > min_cost then
      block()
  let FinalCheck() =
    if cost < min_cost then
      min_cost <- cost
    block()
true
let _ = th.NewAssignment <- (fun p vl -> Assign p vl)
let _ = th.FinalCheck <- (fun () -> FinalCheck())
let _ = th.Pop <- (fun () -> trail.Pop())
let _ = th.Push <- (fun () -> trail.Push())
```
Partial Orders & Object Hierarchies

Acyclic graphs and SMT
Partial Orders as Acyclic Graphs

∀x. x ≤ x
∀x, y. x ≤ y ∧ y ≤ x → x = y
∀x, y, z. x ≤ y ∧ y ≤ z → x ≤ z

Elements are equal in strongly connected components
Partial Orders as Acyclic Graphs

Checking negations

∀x. x ≼ x
∀x, y. x ≼ y ∧ y ≼ x → x = y
∀x, y, z. x ≼ y ∧ y ≼ z → x ≼ z
Partial Orders as Acyclic Graphs

Checking Consistency of $\neg(x \leq y)$:

Is there a $\leq$ path from $\bullet$ to $\bullet$?

Extracting Equalities from $\leq$ using strongly connected components:

```plaintext
let FinalCheck() =  
  for t in terms do   // connect equal terms.  
    let tr = th.GetEqcRoot t  
    if tr <> t then  
      add_eq t tr  
    done  
  check_not_leqs()   // check negations  
  add_implied_eqs()  // propagate equalities  
  true

let NewAssignment (a:Term) v =  
  assert (is_leq a)  
  let args = a.GetAppArgs()  
  let t1, t2 = args.[0], args.[1]  
  add_term t1; add_term t2  
  if v then  
    g.AddEdge t1 t2  
  else  
    trail.Add (fun () -> not_leqs <- List.tail not_leqs)  
    not_leqs <- (t1,t2)::not_leqs

let initialize() =  
  th.FinalCheck <- (fun () -> FinalCheck())  
  th.NewAssignment <- (fun a v -> NewAssignment a v)  
  th.Push <- (fun () -> trail.Push())  
  th.Pop <- (fun () -> trail.Pop())
```
Inheritance as table-lookup

\[ x \preceq \text{java.lang.Comparable} \]

\[ x \preceq \text{java.lang.Cloneable} \]

\[ x = \text{java.util.Date} \]

Efficient propagators using \textit{Type Slicing} algorithm
Leverages ordering of children
J. Gil and Y. Zibin.\[\text{TOPLAS 2007}\]

Available as F#\text{/}Z3 sample

Sherman, Garvin, Dwyer. \textit{IJCAR 2010}
Object Graphs

To Cycle *and* not to Cycle

Intro
SMT?
Z3?

Lazy Reduction
Eager Reduction
Theory Solver

New Theory
Compile
Search
Reduction
New Theory

Constraints
Model
Partial Compile
Saturation

Z3

from Pex
A Theory of Objects

class O {
    public readonly D d;
    public readonly O left;
    public O right;
    public O(D data,
        O left,
        O right) {
        this.data = data;
        this.left = left;
        this.right = right;
    }
}

void f(O n0) {
    Assert (n0 == null ||
        n0.left != n0);
    O n1 = new O(1,null,null);
    O n2 = new O(2,n1,null);
    O n3 = new O(2,n1,null);
    Assert(n2 != n3);
    n1.right = n2;
    n2.right = n3;
    //...
}

Read-only fields
Heap can be updated
Objects are non-extensional
So far so good, but what about read-only fields?

A Theory of Objects

sorts: \( O \),

constructors: \( \text{null} : O, O : H \times D \times O \times O \rightarrow H \times O \),

accessors: \( \text{data} : H \times O \rightarrow D, \text{left} : H \times O \rightarrow O, \text{right} : H \times O \rightarrow O \),

modifiers: \( \text{update\_right} : H \times O \times O \rightarrow H \)

\[
(h', o) = O(h, d, l, r) \implies o \neq \text{null}
\]
\[
(h', o) = O(h, d, l, r) \implies \text{data}(h', o) = d
\]
\[
(h', o) = O(h, d, l, r) \implies \text{left}(h', o) = l
\]
\[
(h', o) = O(h, d, l, r) \implies \text{right}(h', o) = r
\]

\( o \neq \text{null} \implies \text{left}(h_1, \text{left}(h_2, \text{left}(\ldots \text{left}(h_n, o)))) \neq o \)

\[
h' = \text{update\_right}(h, o, r) \land o' \neq o \implies \text{right}(h', o') = \text{right}(h, o')
\]
\[
h' = \text{update\_right}(h, o, r) \implies \text{left}(h', o') = \text{left}(h, o')
\]
\[
h' = \text{update\_right}(h, o, r) \implies \text{data}(h', o') = \text{data}(h, o')
\]
Encoding: Heaps as Arrays

Domains: objects are Natural numbers, left child is a smaller number

\[
O = \mathbb{N} \\
H = \langle \text{data} : O \Rightarrow D, \text{left} : O \Rightarrow O, \text{right} : O \Rightarrow O, \text{clock} : \mathbb{N} \rangle
\]

Most axioms follow by function definitions.

\[
\text{right}(\langle \text{data}, \text{left}, \text{right}, \text{clock} \rangle, o) = \text{select}(\text{right}, o) \\
\text{update_right}(\langle \text{data}, \text{left}, \text{right}, \text{clock} \rangle, o, r) = \langle \text{data}, \text{left}, \text{store}(\text{right}, o, r), \text{clock} \rangle
\]

Only Axiom: Instantiate for every occurrence of left(h,o)

\[
\forall h : H, o : O . \ o \neq \text{null} \implies 0 \leq \text{left}(h, o) < o
\]
Domains: read-only fields use algebraic data-types

\[ O = \text{null} \mid O(id : N, data : D, left : O) \]
\[ H = \langle \text{right} : O \Rightarrow O, \text{clock} : N \rangle \]

Most axioms follow by function definitions.

\[ \text{left}(h, O(id, d, l)) = l \]
\[ \text{left}(h, \text{null}) = \text{null} \]
\[ \text{right}(\langle \text{right}, \text{clock} \rangle, o) = \text{select}(\text{right}, o) \]
\[ \text{update\_right}(\langle \text{right}, \text{clock} \rangle, o, r) = \langle \text{store}(\text{right}, o, r), \text{clock} \rangle \]

No Extra Axiom: Data-type theory enforces acyclicity over \text{left}

\[ \Rightarrow \text{More efficient search} \]
Z3 at the service of $\Gamma, \Pi, \Sigma, \alpha, \beta, \lambda, \eta, \kappa, *, \Box$

SMT version of Satalax, Brown, CADE 2011
Types and Z3 do mingle

Sledge Hammer

Liquid Types

LeonOnline

F* Source: core-ML with dependent refinement types
x: int -> {y: int | x > y}

Preserve types in .NET
class C{a::int => *}

Interop with
C#, VB.NET, F#,
Run on Azure,
Windows Phone 7

F* source

Z3 Type-checker + Compiler

Java Script

rDCIL

DCL

.NET Virtual Machine

Z3

DCL Type-Checker

SMT-LIB2 file

SMT proof witness

parser

parser + preprocessor

Armand, Grégoire, Keller, Théry, Werner
But

Used for First-Order Theorems
Sure, often HOL (problem) is just FO (solution) in disguise

“We are all faced with a series of great opportunities brilliantly disguised as unsolvable problems.”
John W. Gardner

“For every problem there is a solution which is simple, clean and wrong.”
Henry Louis Mencken
Digression: CAL

CAL – Combinatory Array Logic

\[ \text{store}(a, i, v) = \lambda j. \text{if } i = j \text{ then } v \text{ else } a[j] \]

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

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

Existential fragment is in NP by reduction to congruence closure using polynomial set of instances.
but can we do something more HOLish?

e.g.,

\[ \forall f. (\forall x, y. f(x) = f(y) \rightarrow x = y) \rightarrow \exists g. \forall x. x = g(f(x)) \]
Idea: Saturate for Henkin Models

Types
\[ \sigma ::= i \mid o \quad \tau ::= \sigma \mid \tau \rightarrow \tau \]

Terms
\[ M, N ::= \lambda x: \tau . M \mid (M \ N) \mid x \]
\[ \text{false} : o \quad \Rightarrow : o \rightarrow o \rightarrow o \]

Constants
\[ \epsilon : (\tau \rightarrow o) \rightarrow \tau, \quad \forall : (\tau \rightarrow o) \rightarrow o, \]
\[ = : \tau \rightarrow \tau \rightarrow o \]

Axioms
\[ (\forall (\lambda x : \tau . \neg (M \ x))) \lor (M \ (\epsilon M)) \quad \text{for every } M : \tau \rightarrow o \]
\[ M = N \iff (\forall \lambda x : \tau . (M \ x) = (N \ x)) \quad \text{for every } M, N : \tau \rightarrow \tau' \]
\[ (\forall M) \implies (M \ N) \quad \text{for every } M : \tau \rightarrow o, N : \tau \]

Lazy Saturation loop

HOL formula $F$

- **Assert $[F]$**
- **Check SAT**
- **Instantiatiate**

$F \leftarrow F \land F_{\text{Inst}}$

$(\forall (\lambda x : \tau . \neg (M x))) \lor (M (\in M))$

$M = N \iff (\forall \lambda x : \tau . (M x) = (N x))$

$(\forall M) \Rightarrow (M N)$
Propositional reasoning

Equalities

Congruence

Closure

Extensional arrays

\[ \text{HOL} \Rightarrow \text{SMT} \]

\[
\begin{align*}
[\forall M] &= [\forall M] \\
[\varepsilon M] &= [\varepsilon M] \\
[M \implies N] &= [M] \implies [N] \\
[M = N] &= [M] = [N] \\
[(M \cdot N)] &= \text{select}([M], [N]) \\
[\lambda x : \tau . M] &= [\lambda x : \tau . M] \\
[f] &= f \quad \text{for constant } f
\end{align*}
\]
Set of $\beta\eta$ long NF terms with free variables from $\Gamma$ of type $\tau$

Enumerate $T[\Gamma; \tau]$ by depth:

\[(\lambda x : \tau . M) \in T[\Gamma; \tau \rightarrow \tau'] \quad \text{if} \quad M \in T[\Gamma, x : \tau; \tau']\]

\[(x \ M_1 \ldots \ M_k) \in T[\Gamma; \sigma] \quad \text{if} \quad (x : \bar{\tau} \rightarrow \sigma) \in \Gamma, \ M_i \in T[\Gamma; \tau_i]\]

Many more algorithms (matching, unification)/optimizations required for anything viable…

… but main task of Boolean search, equalities, functions is delegated
We surveyed three methods for adding new theories (logics) to Z3:

- As 1\textsuperscript{st} class Theory Solver
- Eager reduction: embed theory in Z3
- Lazy reduction: add facts on demand

Choose one that fits your theory!

[Stan Rosenberg, Anindya Banerjee and David Naumann. Decision Procedures for Region Logic. VMCAI 2012]
Applications often generate problems with particular characteristics (many ground clauses/bit-vectors + predicates/arithmetic + transcendentals/..)

New Z3 feature by de Moura & Passmore:
- Compose strategies using tactical interface.