The Strategy Challenge in SMT Solving (part I)
IWS 2012, Manchester, UK

Leonardo de Moura (Microsoft Research) and Grant Passmore (University of Cambridge)
A Satisfiability Checker with built-in support for useful theories
Satisfiability Modulo Theories (SMT)

\[ b + 2 = c \quad \text{and} \quad f(\text{read(write}(a,b,3), c-2)) \neq f(c-b+1) \]
Satisfiability Modulo Theories (SMT)

\[ b + 2 = c \quad \text{and} \quad f(\text{read}(\text{write}(a, b, 3), c - 2)) \neq f(c - b + 1) \]
\[ b + 2 = c \quad \text{and} \quad f(\text{read}(\text{write}(a, b, 3), c-2)) \neq f(c-b+1) \]
\[ b + 2 = c \quad \text{and} \quad f(\text{read}(\text{write}(a, b, 3), c-2)) \neq f(c-b+1) \]
Main challenges

- Scalability (huge formulas)
- Complexity
- Undecidability
- Quantified formulas
- Nonlinear arithmetic
Basic Idea

\[ x \geq 0, \ y = x + 1, \ (y > 2 \lor y < 1) \]

Abstract (aka “naming” atoms)

\[ p_1, \ p_2, \ (p_3 \lor p_4) \]
\[ p_1 \equiv (x \geq 0), \ p_2 \equiv (y = x + 1), \]
\[ p_3 \equiv (y > 2), \ p_4 \equiv (y < 1) \]
Basic Idea

\[ x \geq 0, \ y = x + 1, \ (y > 2 \lor y < 1) \]

Abstract (aka “naming” atoms)

\[ p_1, \ p_2, \ (p_3 \lor p_4) \]

\[ p_1 \equiv (x \geq 0), \ p_2 \equiv (y = x + 1), \]
\[ p_3 \equiv (y > 2), \ p_4 \equiv (y < 1) \]
Basic Idea

\[ x \geq 0, \ y = x + 1, \ (y > 2 \lor y < 1) \]

Abstract (aka "naming" atoms)

\[ p_1, \ p_2, \ (p_3 \lor p_4) \]
\[ p_1 \equiv (x \geq 0), \ p_2 \equiv (y = x + 1), \]
\[ p_3 \equiv (y > 2), \ p_4 \equiv (y < 1) \]

Assignment

\[ p_1, \ p_2, \ \neg p_3, \ p_4 \]
Basic Idea

\( x \geq 0, \; y = x + 1, \; (y > 2 \lor y < 1) \)

Abstract (aka “naming” atoms)

\( p_1, \; p_2, \; (p_3 \lor p_4) \)

\( p_1 \equiv (x \geq 0), \; p_2 \equiv (y = x + 1), \)
\( p_3 \equiv (y > 2), \; p_4 \equiv (y < 1) \)

Assignment

\( p_1, \; p_2, \; \neg p_3, \; p_4 \)

\( x \geq 0, \; y = x + 1, \)
\( \neg(y > 2), \; y < 1 \)
Basic Idea

\[ \begin{align*}
x &\geq 0, \ y = x + 1, \ (y > 2 \lor y < 1) \\
p_1, \ p_2, \ (p_3 \lor p_4) &\equiv (x \geq 0), \ p_2 \equiv (y = x + 1), \\
p_3 &\equiv (y > 2), \ p_4 &\equiv (y < 1) \\
x &\geq 0, \ y = x + 1, \ \neg(y > 2), \ y < 1
\end{align*} \]
Basic Idea

\[ x \geq 0, \ y = x + 1, \ (y > 2 \lor y < 1) \]

Abstract (aka “naming” atoms)

\[ p_1, \ p_2, \ (p_3 \lor p_4) \]
\[ p_1 \equiv (x \geq 0), \ p_2 \equiv (y = x + 1), \]
\[ p_3 \equiv (y > 2), \ p_4 \equiv (y < 1) \]

SAT Solver

Assignment

\[ p_1, \ p_2, \ \neg p_3, \ p_4 \]
\[ x \geq 0, \ y = x + 1, \ \neg(y > 2), \ y < 1 \]

New Lemma

\[ \neg p_1 \lor \neg p_2 \lor \neg p_4 \]

Unsatisfiable

\[ x \geq 0, \ y = x + 1, \ y < 1 \]

Theory Solver
New Lemma

\neg p_1 \lor \neg p_2 \lor \neg p_4

Unsatisfiable

x \geq 0, y = x + 1, y < 1

AKA
Theory conflict

Theory Solver
Orchestrating Decision Engines
Current SMT solvers provide a combination of different engines.
Combining Engines

- DPLL
- Simplex
- Grobner Basis
- Superposition
- Congruence Closure
- Simplification
- KB Completion
- \( \forall \exists \)-elimination
- ...
Configuring SAT/SMT Solvers: “state-of-the-art”

- F
- Theorem Prover/Satisfiability Checker
- Satisfiable (model)
- Unsatisfiable (proof)

Z3 has approx. 300 options

Config
Actual feedback provided by Z3 users:

“Could you send me your CNF converter?”
“I want to implement my own search strategy.”
“I want to include these rewriting rules in Z3.”
“I want to apply a substitution to term \( t \).”
“I want to compute the set of implied equalities.”
To build theoretical and practical tools allowing users to exert strategic control over core heuristic aspects of high performance SMT solvers.
What is a strategy?

Theorem proving as an exercise of combinatorial search

Strategies are adaptations of general search mechanisms which reduce the search space by tailoring its exploration to a particular class of formulas.
The Need for “Strategies”

Different Strategies for Different Domains.
The Need for “Strategies”

Different Strategies for Different Domains.

From timeout to 0.05 secs...
Example in Quantified Bit-Vector Logic (QBVF)

Join work with C. Wintersteiger and Y. Hamadi
FMCAD 2010

QBVF = Quantifiers + Bit-vectors + uninterpreted functions

Hardware Fixpoint Checks.
Given: $I[x]$ and $T[x, x']$

\[\forall x, x' . I[x] \land T^k[x, x'] \rightarrow \exists y, y' . I[y] \land T^{k-1}[y, y']\]

Ranking function synthesis.
Hardware Fixpoint Checks

[Graphs showing performance comparison between QuBE and Z3 on the left, and sKizzo and Z3 on the right.]
Ranking Function Synthesis

[Graphs showing linear relationships between time (seconds) and some function values for QuBE and sKizzo]
Why is Z3 so fast in these benchmarks?

Z3 is using different engines: rewriting, simplification, model checking, SAT, ...

Z3 is using a customized strategy.

We could do it because we have access to the source code.
SMT solvers are collections of little engines.

They should provide access to these engines. Users should be able to define their own strategies.
Main inspiration: LCF-approach

Tactic

Proof builder

goal

subgoals
Main inspiration: LCF-approach

Proofs for subgoals

Tactic

Proof builder

Goal

Subgoals

Proof for goal

Proofs for subgoals
Main inspiration: LCF-approach
Main inspiration: LCF-approach
Main inspiration: LCF-approach

Proof Builder

Proof Builder

Proof Builder

Proof Builder

thm in LCF terminology

proof in LCF terminology
Tacticals aka Combinators

then(Tactic, Tactic) = Tactic

orelse(Tactic, Tactic) = Tactic

repeat(Tactic) = Tactic
SMT Tactic

goal → Tactic → subgoals

Proof builder
Model builder
goal = formula sequence × attribute sequence

proofconv = proof sequence → proof
modelconv = model × nat → model
trt = sat model
| unsat proof
| unknown goal sequence × modelconv × proofconv
| fail

tactic = goal → trt
SMT Tactic

\[
\begin{align*}
goal & = \text{formula sequence} \times \text{attribute sequence} \\
proofconv & = \text{proof sequence} \rightarrow \text{proof} \\
modelconv & = \text{model} \times \text{nat} \rightarrow \text{model} \\
trt & = \text{sat model} \\
& \quad | \quad \text{unsat proof} \\
& \quad | \quad \text{unknown goal sequence} \times \text{modelconv} \times \text{proofconv} \\
& \quad | \quad \text{fail} \\
tactic & = \text{goal} \rightarrow \text{trt}
\end{align*}
\]

end-game tactics:
never return unknown(sb, mc, pc)
SMT Tactic

good \quad = \text{formula sequence} \times \text{attribute sequence}

proofconv \quad = \text{proof sequence} \rightarrow \text{proof}
modelconv \quad = \text{model} \times \text{nat} \rightarrow \text{model}
trt \quad = \text{sat} \text{ model}
    \quad \mid \text{unsat} \text{ proof}
    \quad \mid \text{unknown} \text{ goal sequence} \times \text{modelconv} \times \text{proofconv}
    \quad \mid \text{fail}
tactic \quad = \text{goal} \rightarrow \text{trt}

non-branching tactics:
sb is a singleton in unknown(sb, mc, pc)
Trivial goals

Empty goal \([\ ]\) is trivially satisfiable

False goal \([\ ..., false, ...\] \) is trivially unsatisfiable

basic : tactic
[ \( a = b + 1, \ (a < 0 \lor a > 0), \ b > 3 \) ]

**Tactic:**

elim-vars

[ \( (b + 1 < 0 \lor b + 1 > 0), \ b > 3 \) ]
SMT Tactic example

Proof builder

[\[ a = b + 1, (a < 0 \lor a > 0), b > 3 \] ]

Tactic: elim-vars

[\[ (b + 1 < 0 \lor b + 1 > 0), b > 3 \] ]

Model builder

M, M(a) = M(b) + 1

M
SMT Tactic example

\[ a = b + 1, \ (a < 0 \lor a > 0), \ b > 3 \]

Tactic: split-or

Proof builder

[ \ a = b + 1, \ a < 0, \ b > 3 \ ]

Model builder

[ \ a = b + 1, \ a > 0, \ b > 3 \ ]
<table>
<thead>
<tr>
<th>SMT Tactics</th>
</tr>
</thead>
<tbody>
<tr>
<td>simplify</td>
</tr>
<tr>
<td>nnf</td>
</tr>
<tr>
<td>cnf</td>
</tr>
<tr>
<td>tseitin</td>
</tr>
<tr>
<td>lift-if</td>
</tr>
<tr>
<td>bitblast</td>
</tr>
<tr>
<td>gb</td>
</tr>
<tr>
<td>vts</td>
</tr>
<tr>
<td>propagate-bounds</td>
</tr>
<tr>
<td>propagate-values</td>
</tr>
<tr>
<td>split-ineqs</td>
</tr>
<tr>
<td>split-eqs</td>
</tr>
<tr>
<td>rewrite</td>
</tr>
<tr>
<td>p-cad</td>
</tr>
<tr>
<td>sat</td>
</tr>
<tr>
<td>solve-eqs</td>
</tr>
</tbody>
</table>
then : \((\text{tactic} \times \text{tactic}) \rightarrow \text{tactic}\)
then\((t_1, t_2)\) applies \(t_1\) to the given goal and \(t_2\) to every subgoal produced by \(t_1\).

then\* : \((\text{tactic} \times \text{tactic sequence}) \rightarrow \text{tactic}\)
then\*\((t_1, [t_{2_1}, ..., t_{2_n}])\) applies \(t_1\) to the given goal, producing subgoals \(g_1, ..., g_m\).
If \(n \neq m\), the tactic fails. Otherwise, it applies \(t_{2_i}\) to every goal \(g_i\).

orelse : \((\text{tactic} \times \text{tactic}) \rightarrow \text{tactic}\)
orelse\((t_1, t_2)\) first applies \(t_1\) to the given goal, if it fails then returns the result of \(t_2\) applied to the given goal.

par : \((\text{tactic} \times \text{tactic}) \rightarrow \text{tactic}\)
par\((t_1, t_2)\) executes \(t_1\) and \(t_2\) in parallel.
SMT Tacticals

\[ \text{then}(\text{skip}, t) = \text{then}(t, \text{skip}) = t \]

\[ \text{orelse}(\text{fail}, t) = \text{orelse}(t, \text{fail}) = t \]
repeat : tactic → tactic
    Keep applying the given tactic until no subgoal is modified by it.

repeatupto : tactic × nat → tactic
    Keep applying the given tactic until no subgoal is modified by it, or the maximum number of iterations is reached.

tryfor : tactic × seconds → tactic
    tryfor(t, k) returns the value computed by tactic t applied to the given goal if this value is computed within k seconds, otherwise it fails.
http://rise4fun.com/z3/tutorial/strategies  (SMT 2.0)

http://rise4fun.com/z3py/tutorial/strategies  (Python)

Z3Py - strategies

Strategies

1. Introduction
2. Tactics
3. Probes
4. tutorials