New and future features of Z3
MSR Cambridge, Z3 SIG meeting

Z3 Team
Leonardo de Moura,
Nikolaj Bjørner
Christoph Wintersteiger
Z3 is a theorem prover developed at MSR
First version released in 2007
First Clients:
- SPEC#/Boogie, SAGE, PEX
Main Features in 2007
- Lazy architecture: SAT solver + theories
- Linear (real) arithmetic, BV, UF, arrays
- E-Matching (for Quantifiers)
- Push/Pop (incremental solving)
Popularized by SMT solvers such as: Simplify.
Part of SMT-LIB 2.0 standard.

push, assert(F1), push, assert(F2), check, pop, assert(F3), check

Is F1 and F2 Sat?

Is F1 and F3 Sat?
Supported Platforms

- Windows 32 and 64 bit versions
- .NET wrapper (C#, F#, VB)
- Linux 32 and 64 bit versions
- New: Mac OSX (starting at Z3 3.3)
Some Applications @ Microsoft

- VCC
- NModel
- SAGE
- SpecExplorer
- Boogie
- SLayer
- Hyper-V
- HAVOC
- Terminator T-2
- Vigilante
- F7
Many external users

> 15k downloads (Windows version)

New Domains → New Requirements
Many engines

SMT

- DPLL
- Simplex
- Grobner Basis
- \( \forall \exists \)-elimination
- Congruence Closure
- Simplification
- Superposition
- 0-1 solver
Yes, we cannot solve arbitrary problems from the “complexity ladder”.
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

![Graph comparing QuBE and Z3 execution times](image1)

![Graph comparing sKizzo and Z3 execution times](image2)
Ranking Function Synthesis
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 engines. They should provide access to these engines. Users should be able to define their own strategies.
Main inspiration: LCF-approach

Tactic

goal

subgoals

Proof converter
Proofs for subgoals

Main inspiration: LCF-approach

Tactic

Proof converter

Proof for goal

subgoals
Main inspiration: LCF-approach

goal → Tactic → Proof conv → Tactic → Proof conv → Proof conv
Main inspiration: LCF-approach

Proof conv

Proof conv

Proof conv

proof
Main inspiration: LCF-approach

Proof conv

thm in LCF terminology

Proof conv

proof in LCF terminology
Tactics aka Combinators

\[\text{then( Tactic }, \text{ Tactic )} = \text{ Tactic}\]

\[\text{orelse( Tactic }, \text{ Tactic )} = \text{ Tactic}\]

\[\text{repeat( Tactic )} = \text{ Tactic}\]
SMT Tactic

Tactic

goal → Tactic

subgoals

Proof converter

Model converter
SMT Tactic example

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

Tactic: elim-vars

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

Tactic: elim-vars

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

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

\[ M, \ M(a) = M(b) + 1 \]

Proof converter

Model converter
SMT Tactic example

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

Tactic: split-clause

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

Proof converter

Model converter
SMT Tactics

simplify
nnf
cnf
tseitin
lift-if
bitblast
gb
vts

propagate-bounds
propagate-values
split-ineqs
split-eqs
rewrite
p-cad
sat
solve-eqs
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.
then(skip, t) = then(t, skip) = t

orelse(fail, t) = orelse(t, 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.
Probing structural features of formulas.
Under-approximation
unsat answers cannot be trusted

Over-approximation
sat answers cannot be trusted
Under-approximation model finders

Over-approximation proof finders
Under/Over-Approximations

Under-approximation
Example: QF_NIA model finders add bounds to unbounded variables (and blast)

Over-approximation
Example: Boolean abstraction
Tactics: supported features

Models
Proofs
Unsat cores

No support for incremental solving

Workaround:
use “all-purpose” incremental solver with timeout
switch to “domain-specific” tactic if can’t solve in $k$ ms
New Arithmetic Solvers

- Current technology
  - Linear Real: Simplex based
  - Linear Integer: Gomory-Cuts, Branching, Model Finder, 0-1
- Nonlinear: Grobner Basis, Branching, Interval Propagation, Model Finder, VTS
New

- Linear Real: New Simplex based on state-of-the-art techniques from MIP
- Linear Integer: cutsat algorithm (complete)
- Nonlinear: cad-like (complete) algorithm, algebraic numbers $x^5 + 2x + 2$, $[-0.8175, -0.8174]$
New solver combining ideas from:
- Interval Propagation
- Reduction to arithmetic
- Bit-blasting

Main target: verification problems
- No overflows
- No NaN
- Error estimation
No plans to support them in Z3

News:
Ken McMillan built a new interpolating prover on top of Z3
Very good experimental results (FMCAD’11)
Z3 API: an update

- Memory management
  - Old: scoped memory management
  - Hack: Z3_persist_ast
  - New: ref-count
    - Z3_mk_context_rc, Z3_inc_ref, Z3_dec_ref

- Logging
  - Old: Z3_open_log (.z3 file), Z3_trace_to_file (.c file)
  - New: global Z3_open_log
Z3 API: an update

- Multiple Solvers & Tactics
- Many solvers per Context
- Incremental & Non-incremental Solvers
- Solvers based on user-provided Tactics
- We will continue to support the old API
http://rise4fun.com/z3/tutorial/guide

New features:
Models as “functional programs”
Quantifiers and Modeling with Quantifiers
Model Based Quantifier Instantiation
BV NEWS
News about Bit-Vectors

- Bit-blasted formulas are large
  - Sometimes unnecessary

- What we’re looking into
  - Stochastic Local Search
  - Custom Propagators
Stochastic Local Search

- Random model
- Improve locally
  - Bit-flips
  - $\pm 1$
  - Negation
  - Et cetera

- Scoring system
  - Relative Satisfaction

\[
x \leq 15 \quad \land \quad y > 0
\]

- $\leq$ node: $17$ with $99.22\%$
- $>$ node: $0$ with $0\%$

MICROSOFT RESEARCH CAMBRIDGE: OVERVIEW PRESENTATION
First Results

![Graph showing Stochastic Local Search vs. Bit-blasting results](image-url)
Quantification

- MBQI for (UF)BV
  - Complete and efficient method
  - Logic being added to SMT-LIB
    - (set-logic UFBV)

- Finite-state problems
  - Model Checking
  - Synthesis