Z3: An Efficient SMT Solver

MSR Cambridge 2007

Leonardo de Moura and Nikolaj Bjørner

{leonardo, nbjorner}@microsoft.com.

Microsoft Research

Introduction

- *Z3 is a new SMT solver developed at Microsoft Research.*
- It is still under development.
- Version 0.1 is the first external release.
- New external releases are coming soon.
- *It replaced Simplify as the default prover for Spec#/Boogie.*
- It won 4 divisions in SMT-COMP'07.

Satisfiability Modulo Theories (SMT)

- SMT is the problem of determining *satisfiability* of formulas *modulo* background *theories*.
- Examples of background theories:
  - linear arithmetic: $x + y \leq 10$
  - bit-vectors: $x \& (x - 1)$
  - uninterpreted functions: $g(f(x)) = x$
  - arrays: $\mathit{read}(\mathit{write}(a, i, v), j) \neq v$
- Example of formula:

$$x + 2 = y, f(\mathit{read}(\mathit{write}(a, x, 3), y - 2)) \neq f(y - x + 1)$$

Applications

- Extended Static Checking.
  - *Microsoft Spec# and ESP*, ESC/Java
- Predicate Abstraction.
  - *Microsoft SLAM/SDV (device driver verification).*
- Bounded Model Checking (BMC) & k-induction.
- Test-case generation.
  - *Microsoft MUTT.*
- Symbolic Simulation.
- Planning & Scheduling.
- Equivalence checking.

Supported Features

- Linear real and integer arithmetic.
- Fixed-size bit-vectors.
- Uninterpreted Functions.
- Extensional Arrays.
- Quantifiers.
- Model generation.
- Coming soon:
  - Improved support for non linear arithmetic.
  - Improved SAT solver.
  - Improved support for quantifiers.
  - Sets & Transitive Closure.

Architecture

- A modern DPLL-based SAT solver.
- A *core theory solver* that handles equalities and uninterpreted functions.
- *Satellite solvers* (for arithmetic, arrays, etc).
- An *E-matching abstract machine* (for quantifiers).
- Very modular: new theories can be added without modifying the core.

Z3: new features

- Model based theory combination.
- E-matching abstract machine.
- “Don’t care” propagation.
- Create/Delete clauses/atoms/terms during backtracking search.

Theory combination

- Theory Combination Problem.
  - $T = T_1 \cup T_2$
  - Is T consistent?
  - Given decision procedures for T_1 and T_2, how to decide T?
- Nelson-Oppen Approach:
  - Disjoint theories.
  - Each theory propagate (disjunction) of implied equalities.
  - For convex theories only implied equalities need to be propagated.

Combining theories in practice

- *Propagate all implied equalities.*
  - Deterministic Nelson-Oppen.
  - Complete only for convex theories.
  - It may be expensive for some theories.
- *Delayed Theory Combination.*
  - Nondeterministic Nelson-Oppen.
  - Create set of interface equalities ($x = y$) between shared variables.
  - Use SAT solver to guess the partition.
  - Disadvantage: the number of additional equality literals is quadratic in the number of shared variables.

Model based theory combination

- Common to these methods is that they are *pessimistic* about which equalities are propagated.
- *Model-based Theory Combination*
  - *Optimistic approach.*
  - Use a candidate model M_i for one of the theories $\mathcal{T}_i$ and propagate all equalities implied by the candidate model, hedging that other theories will agree.

    if $M_i \models \mathcal{T}_i \cup \Gamma_i \cup \{u = v\}$ **then** propagate $u = v$.
  - If not, use backtracking to fix the model.
  - It is cheaper to enumerate equalities that are implied in a particular model than of all models.

Model based theory combination: Example

$$x = f(y - 1), f(x) \neq f(y), 0 \leq x \leq 1, 0 \leq y \leq 1$$

Purifying

Model based theory combination: Example

$$x = f(z), f(x) \neq f(y), 0 \leq x \leq 1, 0 \leq y \leq 1, z = y - 1$$

Model based theory combination: Example

$\mathcal{T}_{\mathcal{E}}$			$\mathcal{T}_{\mathcal{A}}$	
Literals	*Eq. Classes*	*Model*	*Literals*	*Model*
$x = f(z)$	$\{x, f(z)\}$	$x^{\mathcal{E}} = *_1$	$0 \leq x \leq 1$	$x^{\mathcal{A}} = 0$
$f(x) \neq f(y)$	$\{y\}$	$y^{\mathcal{E}} = *_2$	$0 \leq y \leq 1$	$y^{\mathcal{A}} = 0$
	$\{z\}$	$z^{\mathcal{E}} = *_3$	$z = y - 1$	$z^{\mathcal{A}} = -1$
	$\{f(x)\}$	$f^{\mathcal{E}} = \{*_1 \mapsto *_4,$		
	$\{f(y)\}$	$*_2 \mapsto *_5,$		
		else $\mapsto *_1,$		

Assume x = y

Model based theory combination: Example

$\mathcal{T}_{\mathcal{E}}$			$\mathcal{T}_{\mathcal{A}}$	
Literals	*Eq. Classes*	*Model*	*Literals*	*Model*
$x = f(z)$	$\{x, y, f(z)\}$	$x^{\mathcal{E}} = *_1$	$0 \leq x \leq 1$	$x^{\mathcal{A}} = 0$
$f(x) \neq f(y)$	$\{z\}$	$y^{\mathcal{E}} = *_1$	$0 \leq y \leq 1$	$y^{\mathcal{A}} = 0$
$x = y$	$\{f(x), f(y)\}$	$z^{\mathcal{E}} = *_2$	$z = y - 1$	$z^{\mathcal{A}} = -1$
		$f^{\mathcal{E}} = \{*_1 \mapsto *_3,$	$x = y$	
		else $\mapsto *_1\}$		

Unsatisfiable

Model based theory combination: Example

$\mathcal{T}_{\mathcal{E}}$			$\mathcal{T}_{\mathcal{A}}$	
Literals	*Eq. Classes*	*Model*	*Literals*	*Model*
$x = f(z)$	$\{x, f(z)\}$	$x^{\mathcal{E}} = *_1$	$0 \leq x \leq 1$	$x^{\mathcal{A}} = 0$
$f(x) \neq f(y)$	$\{y\}$	$y^{\mathcal{E}} = *_2$	$0 \leq y \leq 1$	$y^{\mathcal{A}} = 0$
$x \neq y$	$\{z\}$	$z^{\mathcal{E}} = *_3$	$z = y - 1$	$z^{\mathcal{A}} = -1$
	$\{f(x)\}$	$f^{\mathcal{E}} = \{*_1 \mapsto *_4,$	$x \neq y$	
	$\{f(y)\}$	$*_2 \mapsto *_5,$		
		else $\mapsto *_1\}$		

Backtrack, and assert $x \neq y$.

$\mathcal{T}_{\mathcal{A}}$ *model need to be fixed.*

Model based theory combination: Example

$\mathcal{T}_{\mathcal{E}}$			$\mathcal{T}_{\mathcal{A}}$	
Literals	*Eq. Classes*	*Model*	*Literals*	*Model*
$x = f(z)$	$\{x, f(z)\}$	$x^{\mathcal{E}} = *_1$	$0 \leq x \leq 1$	$x^{\mathcal{A}} = 0$
$f(x) \neq f(y)$	$\{y\}$	$y^{\mathcal{E}} = *_2$	$0 \leq y \leq 1$	$y^{\mathcal{A}} = 1$
$x \neq y$	$\{z\}$	$z^{\mathcal{E}} = *_3$	$z = y - 1$	$z^{\mathcal{A}} = 0$
	$\{f(x)\}$	$f^{\mathcal{E}} = \{*_1 \mapsto *_4,$	$x \neq y$	
	$\{f(y)\}$	$*_2 \mapsto *_5,$		
		else $\mapsto *_1\}$		

Assume x = z

Model based theory combination: Example

$\mathcal{T}_{\mathcal{E}}$			$\mathcal{T}_{\mathcal{A}}$	
Literals	*Eq. Classes*	*Model*	*Literals*	*Model*
$x = f(z)$	$\{x, z, f(x), f(z)\}$	$x^{\mathcal{E}} = *_1$	$0 \leq x \leq 1$	$x^{\mathcal{A}} = 0$
$f(x) \neq f(y)$	$\{y\}$	$y^{\mathcal{E}} = *_2$	$0 \leq y \leq 1$	$y^{\mathcal{A}} = 1$
$x \neq y$	$\{f(y)\}$	$z^{\mathcal{E}} = *_1$	$z = y - 1$	$z^{\mathcal{A}} = 0$
$x = z$		$f^{\mathcal{E}} = \{*_1 \mapsto *_1,$	$x \neq y$	
		else $\mapsto *_3,$	$x = z$	

Satisfiable

Experimental Results

	#	MathSAT	MathSAT-dtc	Yices	Z3
EufLaArithmetic	52	1851.50 (11)	785.87 (1)	**10.45**	17.34
Hash	199	520.90	19.39	11.48	**6.54**
Wisa	256	886.36 (1)	6916.18	4.37	**2.78**
RandomCoupled	400	517.05	518.15	9516.11 (51)	**56.16**
RandomDecoupled	500	11989.60 (1)	97.07	19362.40 (51)	**41.95**
Simple	98	1366.33	7053.98 (29)	2328.63 (53)	**1.00**
Ackermann	99	228.49 (82)	344.00 (82)	2.99	**1.72**
Total	1604	17360.23 (95)	15734.64 (112)	31236.43 (155)	**127.49**

Quantifiers

- Since first-order logic is undecidable, satisfiability is not solvable for arbitrary quantified formulas.
- Some theories, e.g., datatypes, linear arithmetic over integers, arithmetic over reals, support quantifier elimination.
- Existential quantifiers can be skolemized, but the problem of instantiating universal quantifiers for detecting unsatisfiability remains.

Heuristic Quantifier Instantiation

- Semantically, $\forall x_1, \ldots, x_n.F$ is equivalent to the infinite conjunction $\bigwedge_\beta \beta(F)$.
- Solvers use heuristics to select from this infinite conjunction those instances that are “relevant”.
- The key idea is to treat an instance $\beta(F)$ as relevant whenever it contains enough terms that are represented in the solver state.
- Non ground terms p from F are selected as *patterns*.
- *E-matching* (matching modulo equalities) is used to find instances of the patterns.
- Example: $f(a, b)$ matches the pattern $f(g(x), x)$ if $a = g(b)$.

E-matching

- E-matching is NP-hard.
- The number of matches can be exponential.
- It is not refutationally complete.
- In practice:
  - Indexing techniques for fast retrieval.
  - Incremental E-matching.

E-matching: example

- $\forall x. f(g(x)) = x$
- Pattern: $f(g(x))$
- Atoms: $a = g(b), b = c, f(a) \neq c$
- $\rightarrow$ *instantiate* $f(g(b)) = b$

Quantifiers in Z3

- Z3 uses a E-matching abstract machine.
  - Patterns $\rightsquigarrow$ code sequence.
  - Abstract machine executes the code.
- *Z3 uses new algorithms that identify matches on E-graphs incrementally and efficiently.*
  - E-matching code trees.
  - Inverted path index.
- Z3 garbage collects clauses, together with their atoms and terms, that were useless in closing branches.

E-matching code trees

- In practice, there are several similar patterns.
- *Idea: combine several code sequences in a code tree.*
- Factor out redundant work.
- Match several patterns simultaneously.
- Saturation based theorem provers use a different kind of code tree to implement:
  - Forward subsumption.
  - Forward demodulation.

Incremental E-matching

- Z3 uses a backtracking search.
- *New terms are created during the search.*
  - A code tree for each function symbol f.

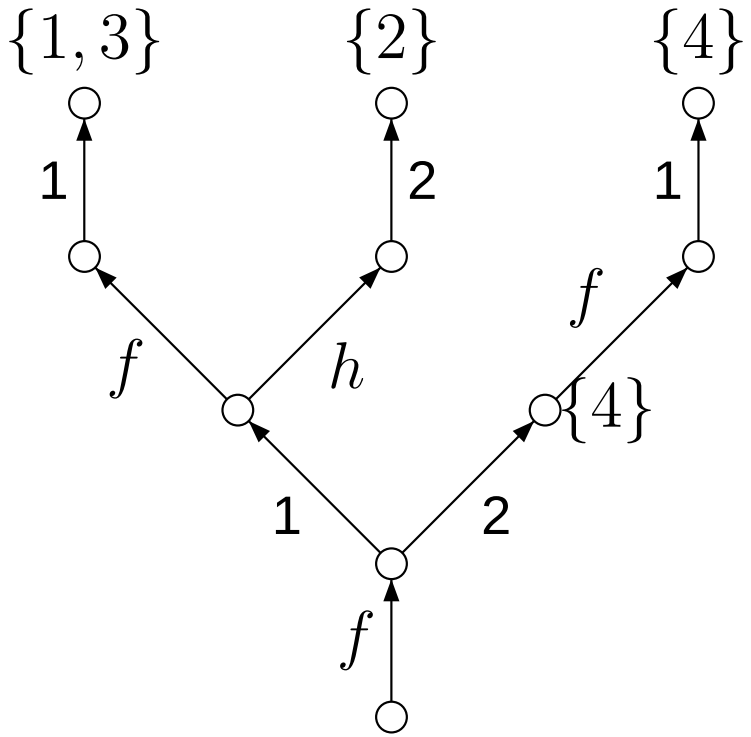
    Patterns that start with a f-application.
  - Execute code-tree for each new term.
- *New equalities are asserted during the search.*
  - New equalities $\rightsquigarrow$ new E-matching instances.
  - Example:

    $f(a, b)$ matches $f(g(x), x)$ after

    $a = g(b)$ is asserted.

Inverted path index

- It is used to find which patterns may have new instances after an equality is asserted.
- Inverted path index for *pc-pair* (f, g) and patterns $f(f(g(x), a), x)$, $h(c, f(g(y), x))$, $f(f(g(x), b), y)$, $f(f(a, g(x)), g(y))$.



E-matching limitations

- E-matching needs ground (seed) terms.
  - It fails to prove simple properties when ground (seed) terms are not available.
  - Example:

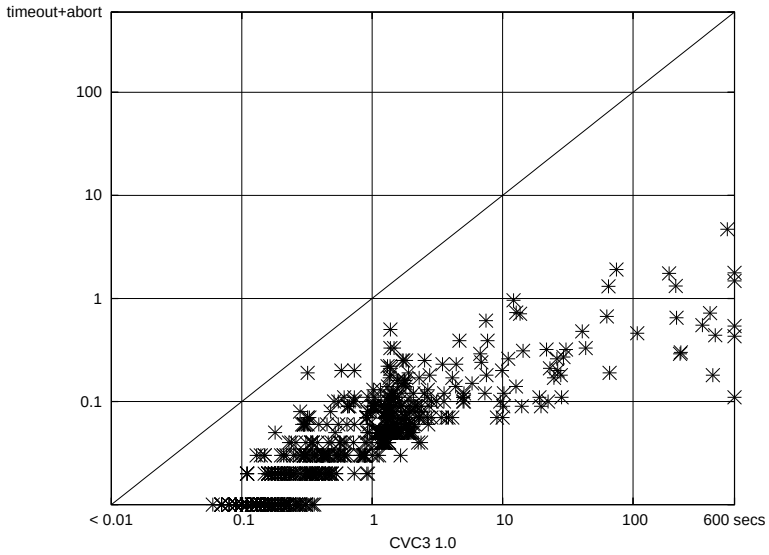
$$(\forall x.f(x) \leq 0) \wedge (\forall x.f(x) > 0)$$

- Matching loops

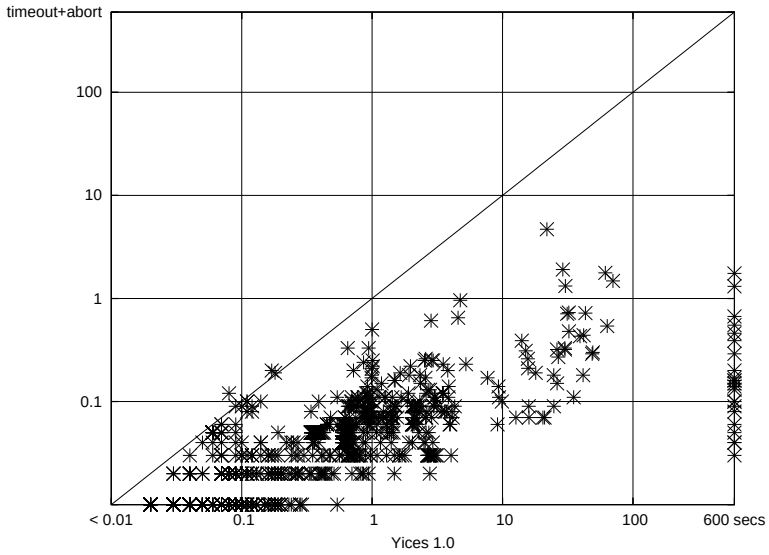
$$(\forall x.f(x) = g(f(x))) \wedge (\forall x.g(x) = f(g(x)))$$

  - Inefficiency and/or non-termination.
  - Some solvers have support for detecting matching loops based on instantiation chain length.

Experimental Results: SMT-LIB

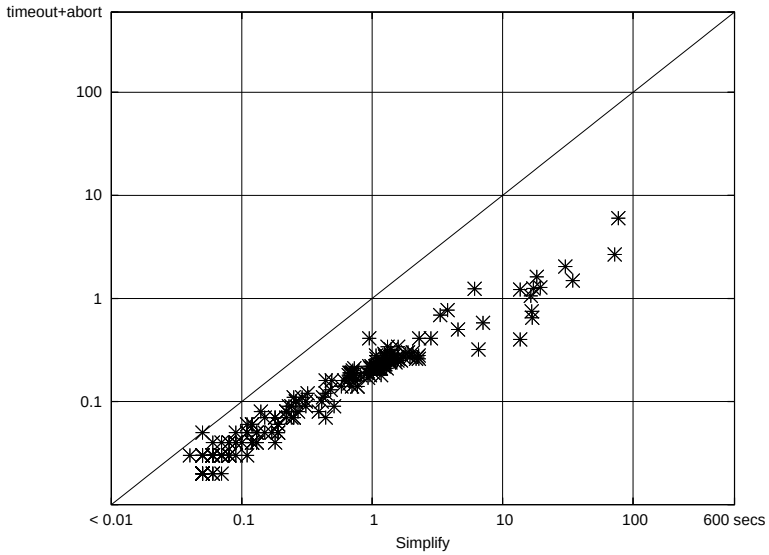


() Z3 vs. CVC3 1.0

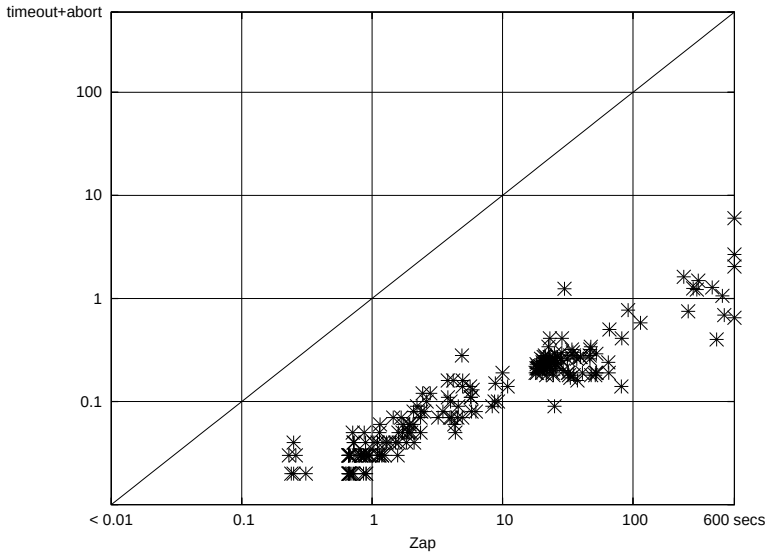


() Z3 vs. Yices 1.0

Experimental Results: ESC/Java

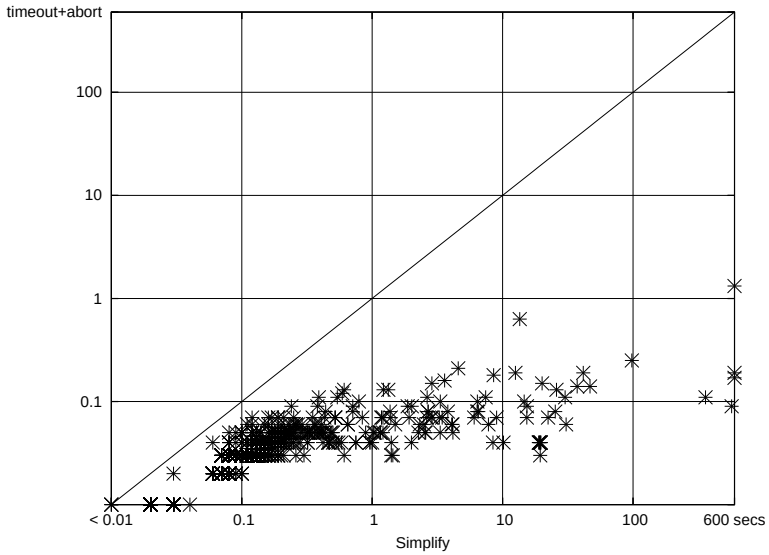


() Z3 vs. Simplify

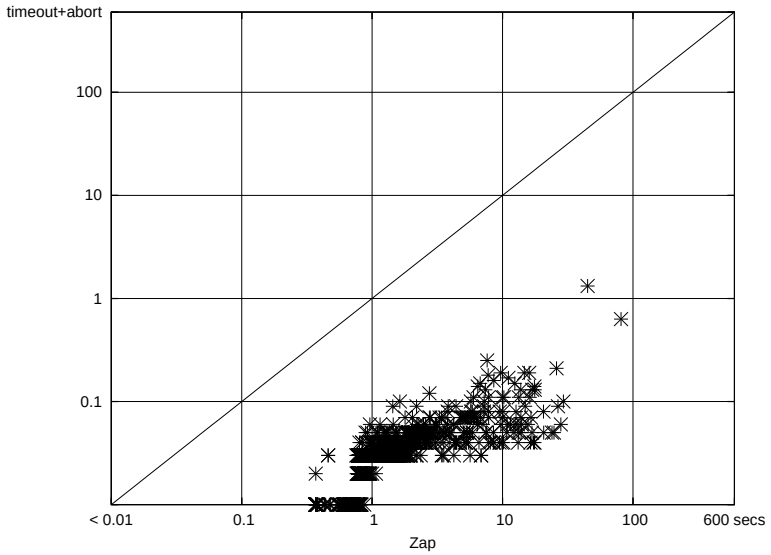


() Z3 vs. Zap 2.0

Experimental Results: Boogie

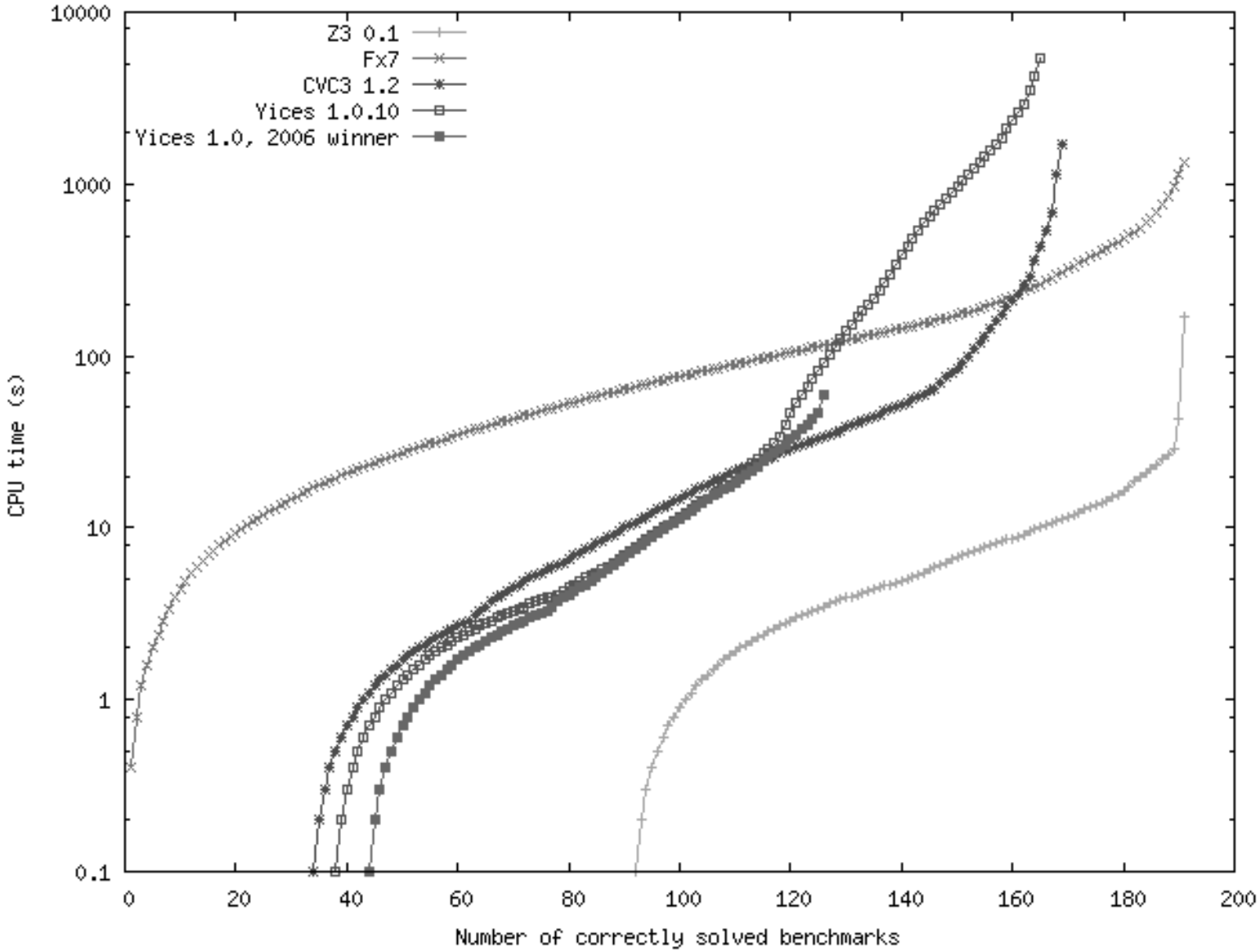


() Z3 vs. Simplify



() Z3 vs. Zap 2.0

SMT-COMP'07: AUFLIA



Don’t cares

- DPLL(T) based solvers assign a boolean value to potentially all atoms appearing in a goal.
- In practice, several of these atoms are *don’t cares*.
- *Z3 ignores don’t care atoms for expensive inference rules and theories, such as, quantifier instantiation.*

Bit-vector theory

- Simple approach:
  - Based on bit-blasting.
  - Partial evaluation.
  - “Don’t cares”.
- *Supports all operations used in software and hardware.*
- They can be used with other theories and quantifiers.
- *Z3 uses an efficient and modular pre-processor.*

Getting Z3

- Internal
  - Stable builds: `http://codebox/z3`
  - Fresh builds: \\pexbuild3\drops\z3\latest\release
- External
  - Official website: `http://research.microsoft.com/projects/z3`
  - SMT-COMP'07: `http://www.smtcomp.org`
- Managed & Unmanaged APIs.
- SMT-LIB & Simplify input formats.

Current (and future) users

- Boogie/Spec#
- SLAM/SDV
- Pex
- Vigilante
- *Your project?*

Future work

- Improved support for quantifiers:
  - Model checking.
  - Superposition calculus + SMT.
  - Decidable fragments.
- Machine learning & dynamic tuning.
- Transitive Closure.
- Improved bit-vector and array theories.
- Better performance.
- More “customers” & applications.

Conclusion

- Z3 is an efficient and modular SMT solver.
- It is going to be used in several projects at Microsoft.
- It can solve 99.7% of the benchmarks in SMT-LIB.
- New releases of Z3 will be available at:
  `http://research.microsoft.com/projects/z3`.