Z3str3: A String Solver with Theory-Aware Heuristics

Murphy Berzish [1], Yunhui Zheng [2], **Vijay Ganesh** [1]

[1] University of Waterloo

[2] IBM Research

UNIVERSITY OF WATERLOO

Outline

- Background and overview
- The Z3str3 string solver
- New heuristics
  - Theory-aware branching
  - Theory-aware case split optimization
- Experimental results
- Future work and conclusions

Overview

- String SMT solvers increasingly used for security applications and analysis of string-intensive programs

- Many tools developed to address these challenges and applications: Z3str2, CVC4, Norn, S3, Stranger

- Need for more efficient solvers and heuristics: complex semantics, easy to create undecidable theories, crossover with strings and other theories (arithmetic, bit-vector)

Known Theoretical Results

- In 1946, Quine showed that the fully-quantified theory of word equations is undecidable
- In 1940's Markov suggested using word equations to settle Hilbert's Tenth Problem
- In 1968, Matiyasevich showed a reduction from word equations+length to Diophantine
- In 1977, Makanin showed that the quantifier-free theory of word equations is decidable
- In 2012, word equations with single quantifier-alternation was shown to be undecidable [GRSM 2012]
- In 2016, word equations, length, string-integer conversion shown undecidable [GB 2016]
- Matiyasevich's challenge remains open

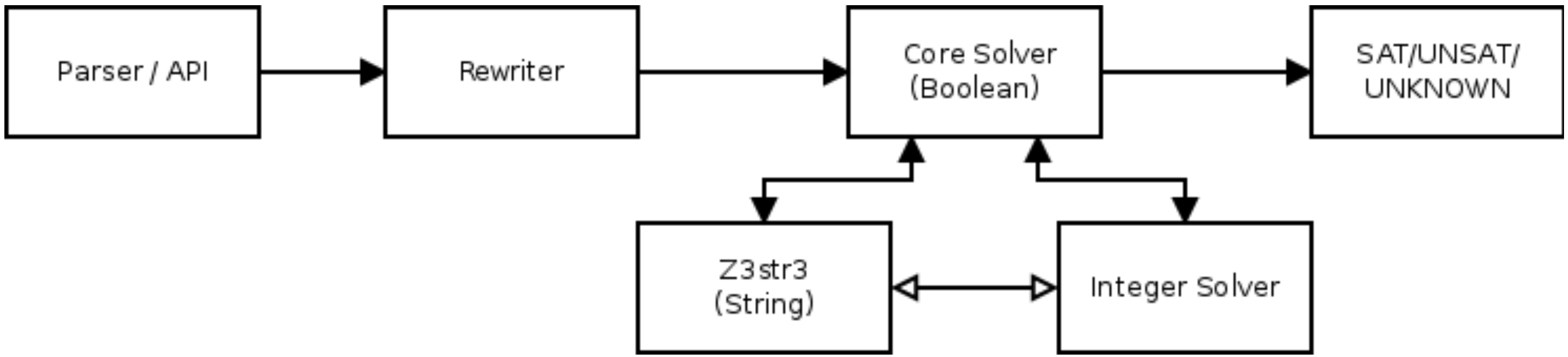
Input Language of Z3str3

String and integer constants	“abc”, “new\nline”, 123
String concatenation	(str.++ “abc” “def”)
String length	(str.len “abcdef”)
Integer arithmetic	(+ 2 2)
String equality	(= X “abc”)
Integer comparison	(= X 42), (<= A 100)
Regular language membership	(str.in.re “aaa” (re.* (str.to.re “a”)))
High-level string operations	(str.prefixof “abc” “abcdef”), (str.contains X “abc”), ...

The Z3str3 String Solver

- Successor to Z3-str and Z3str2
- Native first-class theory solver in Z3 SMT solver framework
- Primary string solver in Z3 official release
- Reasoning about strings, length, regular expressions, and high-level string operations
- **Direct access to the core solver** of Z3 has enabled new heuristics

Architecture of Z3str3

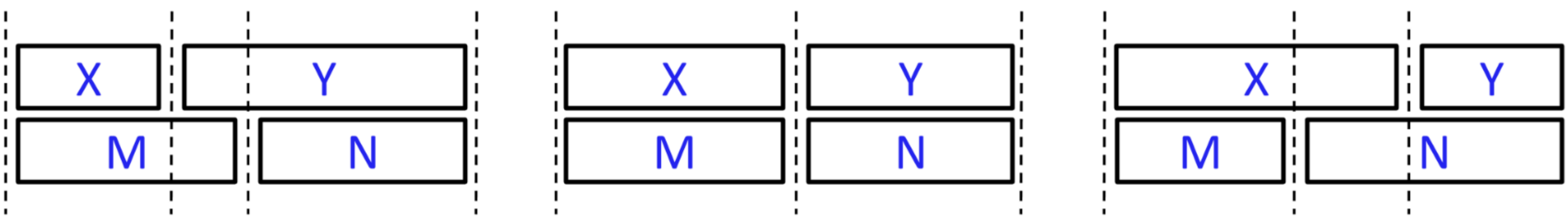


UNIVERSITY OF WATERLOO

How Z3str3 Solves Word Equations

- Given an equality between string terms, identify all possible **arrangements** of subterms
- Generate smaller equations implied by the equality
- Recursively split until the problem is directly solvable

$$X . Y = M . N$$



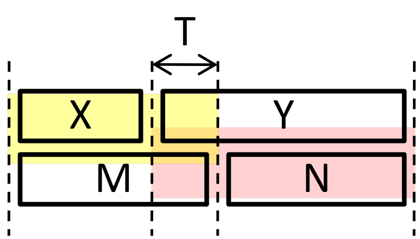
3 possible arrangements

UNIVERSITY OF WATERLOO

Solving String Equations

- **Basic idea**
  - Recursively split equations into smaller ones until they are directly solvable
  - Given an equation, identify all possible arrangements
  - Given an arrangement, generate smaller equations

X . Y = M . N



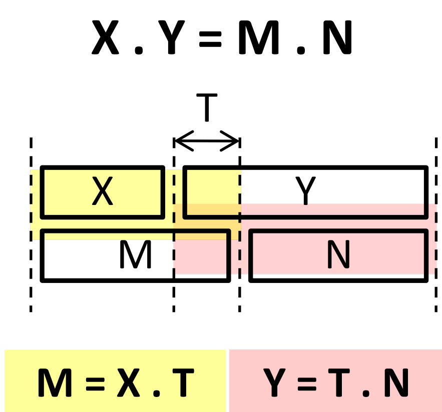
M = X . T | **Y = T . N**

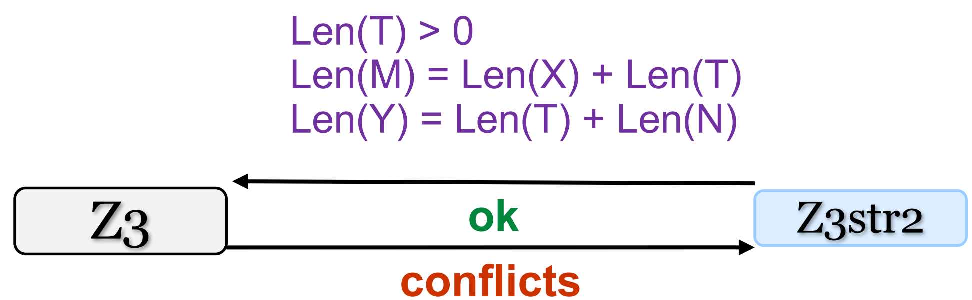
Smaller Equations

- Keep splitting until solved
- If conflicts detected, rollback, try another arrangement

Sync with Integer Theory

- **Consistent solutions in both theories**
  - Z3str2 asserts new length constraints during search





- Keep splitting
- Rollback. Try another arrangement

Theory-Aware Branching

- Traditional DPLL(T) architecture separates core (Boolean) solver from theory solvers
- Theory solvers have contextual information which core solver doesn't know
- Idea: use this to improve performance in core by preferring “easier” or “more important” literals

Theory-Aware Branching

- Activity-based branching heuristic (similar to VSIDS): branch on literal with highest activity
  - Activity increased by conflicts, decays over time

- Theory solvers can increase or decrease activity of literals

- Advantage: give the core solver **information regarding the relative importance of each branch**, allowing the theory solver to **exert additional control over the search**.

Theory-Aware Branching

- Consider the case where the string solver learns

$$X . Y = A . B$$

(for non-constant terms A, B, X, Y)

- The solver considers three possible arrangements:
  - $X = A, Y = B$
  - $X = A . s_1, s_1 . Y = B$ for a fresh non-empty string s_1
  - $X . s_2 = A, Y = s_2 . B$ for a fresh non-empty string s_2
- The first arrangement is the **simplest to check**: no new variables
- Theory solver **adds activity** to the literal corresponding to this arrangement; this prioritizes checking it

Theory-Aware Case Split

- A different way to use information from theory solvers to guide search in the core
- Theory solver can create disjunctions of Boolean literals which are **pairwise mutual exclusive**
- We refer to this as a "theory case split"

Theory-Aware Case Split

- Consider the case where the string solver learns:

  $X . Y = s = c_1 c_2 c_3 \ldots c_n$

  for variables X, Y and where each c_i is a single character in the string constant s

- There are n+1 possible ways in which we can split s over X and Y

- Each arrangement represents a mutually exclusive case

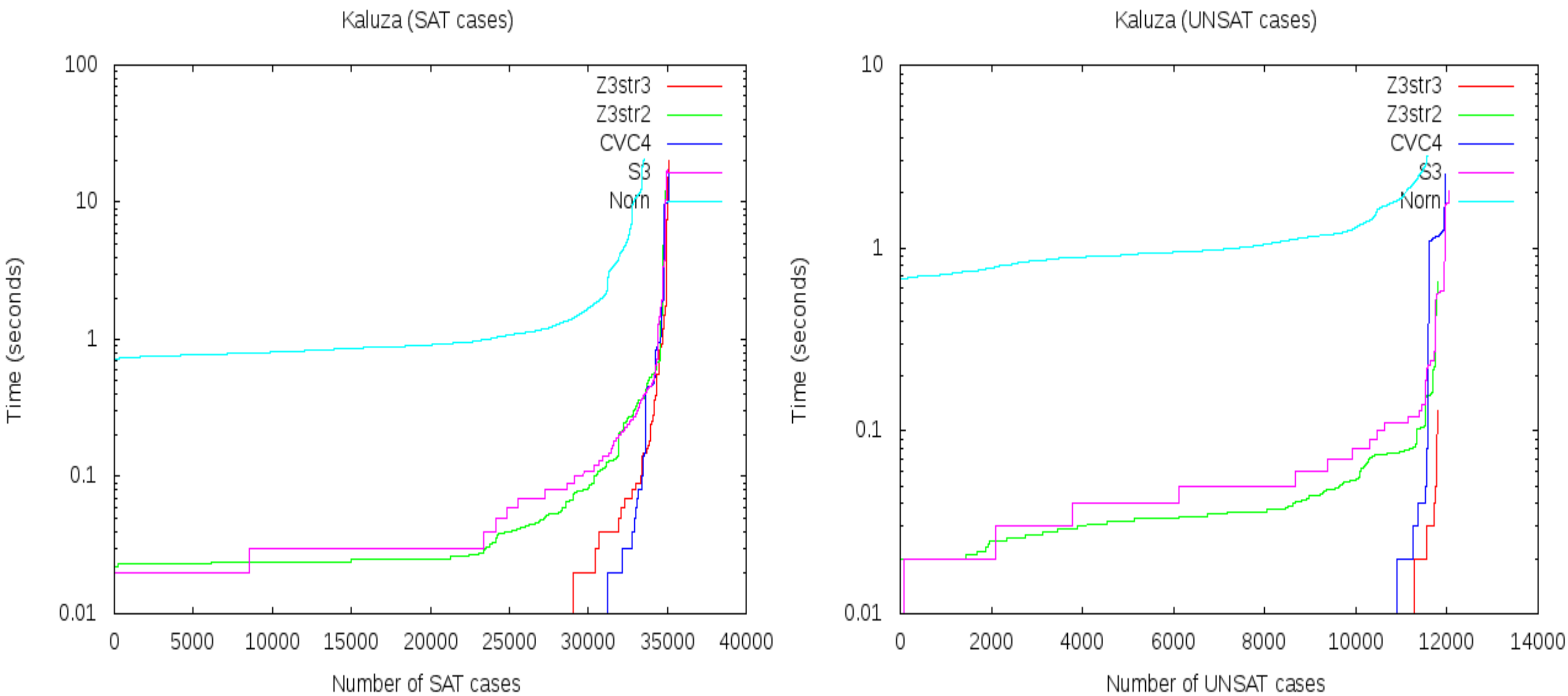
Theory-Aware Case Split

- The Boolean abstraction hides the fact that these are mutually exclusive cases
- Naive solution encodes $O(n^2)$ extra mutual exclusion clauses
- Congruence closure can "discover" this fact, but this can result in unnecessary backtracking
- Previous work has investigated alternate encodings, e.g. totalizers and lazy cardinality
- Our heuristic implements this mutual exclusion **in the inner loop of Z3's core solver in a theory-aware manner**

Theory-Aware Case Split

- Theory solver provides a set S of mutually-exclusive literals to the core solver
- During branching, core solver checks whether the current branching literal is in some set S. If yes, that literal is assigned true and all other literals in S are assigned false.
- During propagation, if the core solver assigns a literal in some set S, the solver must check whether any two literals L_1, L_2 in S have both been assigned true. If so, the core solver generates conflict clause (not L_1 or not L_2)

Experimental Results



Kaluza benchmark results. Timeout = 20 seconds.

UNIVERSITY OF WATERLOO

Experimental Results

Input	Z3str3		Z3str2		CVC4		S3	
	result	time (s)	result	time (s)	result	time (s)	result	time (s)
pisa-000.smt2	sat	0.03	sat	0.25	sat	0.08	sat	0.07
pisa-001.smt2	sat	0.05	sat	0.19	sat	0.00	sat	0.07
pisa-002.smt2	sat	0.03	sat	0.10	sat	0.00	sat	0.05
pisa-003.smt2	unsat	0.02	unsat	0.02	unsat	0.01	unsat	0.02
pisa-004.smt2	unsat	0.02	unsat	0.05	unsat	0.39	unsat	0.05
pisa-005.smt2	sat	0.02	sat	0.14	sat	0.02	sat	0.04
pisa-006.smt2	unsat	0.03	unsat	0.05	unsat	0.32	unsat	0.05
pisa-007.smt2	unsat	0.02	unsat	0.05	unsat	0.37	unsat	0.05
pisa-008.smt2	sat	0.43	timeout	20.00	timeout	20.00	unsat **X**	4.73
pisa-009.smt2	sat	0.60	sat	0.62	sat	0.00	timeout	20.00
pisa-010.smt2	sat	0.02	sat	0.09	sat	0.00	unsat **X**	0.02
pisa-011.smt2	sat	0.03	sat	0.06	sat	0.00	unsat **X**	0.02

PISA benchmark results. Timeout = 20 seconds. **X** = incorrect response.

Experimental Results

Input	Z3str3		Z3str2		CVC4		S3	
	result	time (s)	result	time (s)	result	time (s)	result	time (s)
t01.smt2	sat	0.18	sat	1.31	sat	0.01	sat	0.23
t02.smt2	sat	0.17	sat	0.38	sat	0.01	unknown	0.04
t03.smt2	sat	0.27	sat	9.54	sat	3.82	sat **X**	0.14
t04.smt2	sat	0.73	sat	4.45	timeout	20.00	sat **X**	0.10
t05.smt2	sat	0.57	sat	16.84	sat	3.87	sat **X**	0.55
t06.smt2	sat	0.02	sat	0.15	sat	0.01	sat	0.13
t07.smt2	sat	2.18	sat	0.25	sat	0.00	unknown	0.02
t08.smt2	sat	0.03	sat	0.25	sat	0.17	sat **X**	0.03

IBM AppScan benchmark results. Timeout = 20 seconds. **X** = incorrect response.

Experimental Results

	No heuristics	Theory-aware branching	Theory-aware case split	Both heuristics
sat	35079	35147	35092	35147
unsat	11799	11799	11799	11799
unknown	221	230	223	223
timeout	185	108	170	115
Total time (s)	6252.26	6055.04	5027.35	4939.52

Performance comparison with individual heuristics.
Times taken over Kaluza benchmark. Timeout = 20 seconds.
Total time includes all solved, timeout, and unknown instances.

Future Work

- Improved heuristics for mutually referential terms ("overlapping variables")
- String + bit-vector reasoning
- Regular expression support
- CFG support

Conclusions

- We present the Z3str3 string solver, newest in the Z3-str line
- Primary string solver used by Z3 official release
- Improved performance over predecessor and competitors on majority of industrial benchmarks
- Heuristics are broadly applicable to SMT solvers

https://sites.google.com/site/z3strsolver

https://github.com/Z3prover/Z3