Z3str3: A String Solver with Theory-Aware Heuristics

Murphy Berzish ¹, Yunhui Zheng ², Vijay Ganesh ¹

¹ University of Waterloo
² IBM Research
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

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

Parser / API → Rewriter → Core Solver (Boolean) → SAT/UNSAT/UNKNOWN

Z3str3 (String) → Integer Solver
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

\[ X \cdot Y = M \cdot N \]

3 possible arrangements
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

\[
X \cdot Y = M \cdot N
\]

\[
\begin{array}{c}
M = X \cdot T \\
Y = T \cdot N
\end{array}
\]

- 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

\[ X \cdot Y = M \cdot N \]

- Keep splitting
- Rollback. Try another arrangement

\[ M = X \cdot T \quad Y = T \cdot N \]

Len(T) > 0
Len(M) = Len(X) + Len(T)
Len(Y) = Len(T) + Len(N)
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
  \[ X \cdot Y = A \cdot B \]
  (for non-constant terms \( A, B, X, Y \))

- The solver considers three possible arrangements:
  - \( X = A, Y = B \)
  - \( X = A \cdot s_1, s_1 \cdot Y = B \) for a fresh non-empty string \( s_1 \)
  - \( X \cdot s_2 = A, Y = s_2 \cdot B \) 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:
  \[ X \cdot Y = s = c_1 c_2 c_3 \ldots c_n \]
  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.
## Experimental Results

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

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

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

IBM AppScan benchmark results. Timeout = 20 seconds. X = incorrect response.
## Experimental Results

<table>
<thead>
<tr>
<th></th>
<th>No heuristics</th>
<th>Theory-aware branching</th>
<th>Theory-aware case split</th>
<th>Both heuristics</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>sat</strong></td>
<td>35079</td>
<td>35147</td>
<td>35092</td>
<td>35147</td>
</tr>
<tr>
<td><strong>unsat</strong></td>
<td>11799</td>
<td>11799</td>
<td>11799</td>
<td>11799</td>
</tr>
<tr>
<td><strong>unknown</strong></td>
<td>221</td>
<td>230</td>
<td>223</td>
<td>223</td>
</tr>
<tr>
<td><strong>timeout</strong></td>
<td>185</td>
<td>108</td>
<td>170</td>
<td>115</td>
</tr>
<tr>
<td><strong>Total time (s)</strong></td>
<td>6252.26</td>
<td>6055.04</td>
<td>5027.35</td>
<td>4939.52</td>
</tr>
</tbody>
</table>

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