AST - Progress Report

PASCAL STREBEL and ROBIN SCHMIDIGER

1 PROGRESS SUMMARY

We designed a technique for creating and rewriting (un)satisfiable Satisfiability Modulo Theories (SMT) formulas based on the edit distance [Levenshtein 1965] between strings. We implemented this technique in the tool MadamASTra to test Z3's String solvers Z3Seq [Berzish et al. 2021] and Z3Str3 [Berzish et al. 2017]. In the current state, we have designed and built a software architecture with basic functionalities for doing so, which is still to be extended.

2 METHODOLOGY

The edit distance x between two strings s_1 and s_2 corresponds to the minimum number of inserts, removals and replacements of a single character to transform s_1 into s_2. That is, it is possible to find a sequence of x `insert`/`remove`/`replace` operations to get from $s1$ to $s2$, and it is impossible to find a sequence of $< x$ such operations that achieves the same. Those two thoughts form the basis for our generation of satisfiable and unsatisfiable SMT formulas, respectively.

In terms of SMT solving, we model the `insert`/`remove`/`replace` operations as interpreted functions.[1] For instance, the SMT encoding of the `insert` operation is shown in Figure 1. We then automatically generate formulas including series of applications of these functions. These formulas can include more or less unknowns, as shown in Figures 2 and 3, respectively, and can be used to test the String solvers of Z3.

```
(define-fun insert ((to_insert String) (index Int) (source String)) String
(str.++ (str.substr source 0 index) to_insert (str.substr source index (- (str.len source) index))))
```

Fig. 1. SMT encoding of the `insert` operation. In particular, the character `to_insert` is inserted in the string `source` at position `index`. Note how this definition does not enforce that `to_insert` is a single character, therefore we do this with additional constraints on each application of the function.

```
φ1: (assert (= (remove 4 (insert "p" 2 (replace "w" 0 "host"))) "wops"))
φ2: (assert (= (remove int_const_2 (insert str_const_1 int_const_1 (replace str_const_0 int_const_0 "host"))) "wops"))
```

Fig. 2. Satisfiable SMT formulas generated by our technique. Formula φ_1 contains not only the `insert`/`remove`/`replace` operations to be applied, but also the corresponding input values to get from 'host' to 'wops'. φ_1 is not a challenge for Z3, but an implicit model for the correctness of our technique. In formula φ_2, these values are replaced by uninterpreted constants, i.e., unknowns whose value should be found. The more unknowns there are, the more difficult it becomes for Z3 to determine whether the formula is satisfiable. Note that we must additionally enforce all string constants to have length 1 (via `(assert (= (str.len str_const_x) 1))`).

[1] If we model `insert`/`remove`/`replace` as uninterpreted functions (as initially planned), i.e., not specifying their behavior, Z3 tends to infer a definition containing many if-then-else cases (e.g. if the input for `insert` is `"o" "wrd" 1` then output is `"text"`) that do not reflect the actual specification of the operation. Enforcing the specification requires usage of quantors (e.g. ∀ s : String. ∀ c : Char. ∀ i : Int. len(x)+1 = len(insert(c, i, s))), which introduces further theories and causes the SMT solver to time out in most cases.

Authors' address: Pascal Strebel, pstrebel@ethz.ch; Robin Schmidiger, schmrobi@ethz.ch.

```
φ3: (assert (= (replace str_const_1 int_const_1 (remove int_const_0 "host")) "wops"))
```

Fig. 3. Unsatisfiable formula generated by our technique. Since we know that the edit distance between 'host' and 'wops' is 3, it is impossible to get from 'host' to 'wops' with only one `insert` and one `replace` operation. Thus, φ_3 is unsatisfiable by construction.

3 IMPLEMENTATION

We created MadamASTra, an automated tool incorporating the procedure described in the previous section as a form of metamorphic testing.[2] Algorithm 1 gives a high-level overview of the current pipeline of the software.

In each iteration of the main procedure, we first generate two random words. The longer the words, the higher the difficulty, as the edit distance between them tends to be higher. Next, we compute the edit distance between them and encode it in an SMT-instance that is (un)satisfiable by construction. Although a Python API exists for Z3, we deliberately take the detour with generating an SMT formula so that our technique is applicable for other SMT solvers as well. We then spawn a shell and run Z3 with the SMT-instance. If Z3 makes an error, i.e., erroneously labels the formula as (un)satisfiable, we remember it. This procedure is repeated for a set number of runs and all errors are collected.[3]

Algorithm 1 MadamASTra

```
bugs ← {}
while True do
    w1, w2 ← wordgenerator.generate()
    if mode == "sat" then
        φ ← get_sat_z3_formula(w1, w2)
    else
        φ ← get_unsat_z3_formula(w1, w2)
    end if
    input = wrap_formula(φ, config)
    res = z3_driver.run(input)
    if res! = mode then
        bugs ← bugs ∪ {φ}                    ▷ Z3 made a mistake!
    end if
end while
```

4 TO DO

In the remainder of this project, we will extensively refine MadamASTra and thoroughly verify its correctness before unleashing it on Z3's two string solvers on a larger scale. To make possible error-causing SMT instances usable, we also need to incorporate some form of delta debugging. Moreover, the logging procedure of issues is very simplistic up to now, which we plan to enhance. And finally, the random words are currently generated in a naive way; we aim to instead implement a system where the word length as well as the edit distance can be specified.

REFERENCES

Murphy Berzish, Vijay Ganesh, and Yunhui Zheng. 2017. Z3str3: A String Solver with Theory-aware Heuristics. In *2017 Formal Methods in Computer Aided Design (FMCAD)*. 55–59. https://doi.org/10.23919/FMCAD.2017.8102241

Murphy Berzish, Mitja Kulczynski, Federico Mora, Florin Manea, Joel D. Day, Dirk Nowotka, and Vijay Ganesh. 2021. An SMT Solver for Regular Expressions and Linear Arithmetic over String Length. arXiv:2010.07253 [cs.LO]

V. I. Levenshtein. 1965. Binary codes with correction of deletions, insertions and symbol substitutions. , 845-848 pages.

[2]For the implementation, we use python (3.10.6) and Z3 (4.8.15).
[3]Note that we do not count it if Z3 returns unknown or timeout.