Strings Enhanced Symbolic Execution

Treating Strings as ADTs in a KLEE/Z3 framework

May 11, 2018

Background

- String intensive programs are abundant, and analyzing them naively in a symbolic execution framework is hard: string library code is analyzed as a bundle with application code, spawning a huge amount of irrelevent states.
- String SMT solvers allow a direct encoding of string library code to SMT formulae, thus reducing dramatically the number of explored states.
- Several string solvers exist: Hampi, Kaluza, Pisa, Stranger CVC4, S3, Z3str3, and they have been used (to some extent) to analyze SQL querries and programs in Java, JavaScript and PHP.
- str.KLEE enables symbolic execution of arbitrary C programs, which are (considerably) more widespread than previously mentioned PL, and have a more complicated semantics. It uses KLEE together with Z3, state-of-the-art SE and solver.

Implementation (High Level) Description

- Resolving occurrences of program string variables is not trivial in C, which is weakly typed, and allows taking the memory location of stored variables: msg→contents = (void *) &buf;
- We use the term abstract buffers to denote the corresponding solver string variables. As the program modifies strings with writes, strcpy etc. abstrct buffers accumulate these changes by keeping consecutive versions for abstract buffers.
- Using a many sorted solver inevitably introduces sort conversion issues. Suppose that a bitvector is added to an integer: (i << 5) + strlen(msg) and it is sound to do this addition in both bitvector and integer domains. Which expression should be converted? we say: whichever is faster!

Accelerations List

- Context Aware Sorts
  - ((*s) == 'a') vs. ((*s) << 5)
- Tailored String Semantics
  - int strcmp(s1,s2) $\rightarrow$ bool strcmp(s1,s2)
  - char *strchr(s,c) $\rightarrow$ bool strchr(s,c)
- Solver Performance Driven Query Rewriting
  - str.indexof $\rightarrow$ str.contains
  - str.indexof $\rightarrow$ str.len
  - automatic deducing of query invariants
- Reducing Number of States with C to C translations:
  char *f(char *d,char *s) { while (*d++ = *s++); } return d;
  $\rightarrow$ strcpy(d,s); return d+strlen(src);
- Caching reads/writes.
- Reducing number of generated abtract buffer versions.
- Under approximation of programs paths. For example, Ignoring toUpper and adding relevant constraints.

Accelerations :: Context Aware Sorts

- Sort conversions are expansive, and in some cases they can be avoided altogether. Think of the following examples:

  if ((*s) == ' ') { s++; }

  if (((*s) << 3) < 100) { s++; }

  Since the returned value from (*s) has a string sort, then in order to shift it left, it needs to be converted to a bit vector. In constrast, if it is simply compared to the space character, then it needs not be converted at all.
- A simple pre processing of the program can easily identify locations where such conversions are not needed.

Accelerations :: Tailored String Semantics

- The semantics of string library functions often contains more details then actually needed by users. For example, whenever two strings are compared with strcmp(s1,s2), the returned value is either 0 when they areidentical, or the ascii difference between the first place of change is returned. Almost all users simply ask whether the result is 0 or not. This gives a chance for a sound optimization, since it enables us to use the native returned boolean from the solver, and ignore the ascii difference since the program makes no use of this value.
- Similarly, some programs use strchr to simply check the existence of a certain character in a string:
  if (strchr(s, ' ')) { return -1; }

Accelerations :: Query Rewriting

- The semantics of string library functions often contains more details then actually needed by users. For example, whenever two strings are compared with strcmp(s1,s2), the returned value is either 0 when they areidentical, or the ascii difference between the first place of change is returned. Almost all users simply ask whether the result is 0 or not. This gives a chance for a sound optimization, since it enables us to use the native returned boolean from the solver, and ignore the ascii difference since the program makes no use of this value.
- Similarly, some programs use strchr to simply check the existence of a certain character in a string:
  if (strchr(s, ' ')) { return -1; }