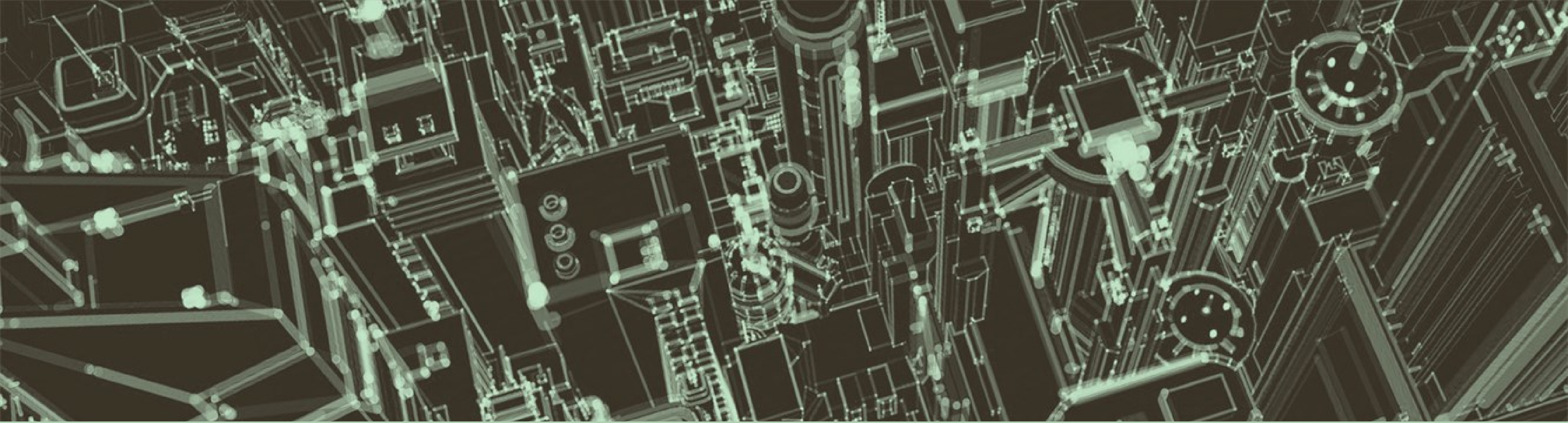
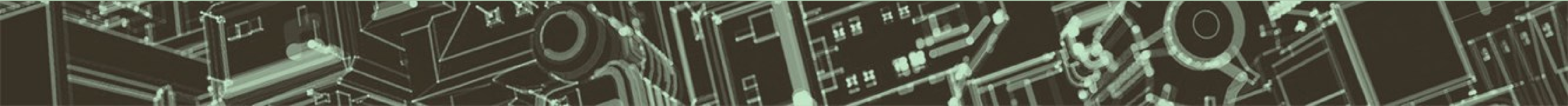


AFL Fuzzing Algorithm

M. Raveendra Kumar (raveendra.kumar@tcs.com)
K.V. Raghavan (raghavan@iisc.ac.in)



Given: Instrumented program P, and a set of valid seed inputs S.

Data structures:

1. $Q = \emptyset$. $\text{currQ} = \text{null}$. // Working queue of test cases
2. $Q_x = \emptyset$ // Queue of crashed and hanging test cases
3. $G = Table\ with\ 2^{16}\ entries,\ each\ entry\ being\ 8\ bits\ (one\ byte), initialized\ to\ 0xFF$ // A zero at k^{th} bit position in a table entry $G[bp]$ indicates that in some run seen so far **roundOff**(visit count of branch-pair bp) = 2^k, where $0 \le k \le 7$. This structure persists across all runs of P.
4. $Shm = Table\ with\ 2^{16}\ entries,\ each\ entry\ being\ one\ byte, initalized\ to\ zero$ // Shared memory table. Shared with the program P. Is re-initialized for each run of P. $Shm[bp]$ contains (rounded off) visit count to branch pair bp by the current run.
5. $R = Table\ with\ 2^{16}\ entries,\ each\ entry\ initalized\ to\ null$ // Each table entry $R[bp]$ contains a preferred test input for that branch pair bp. Persists across all runs.
6. $cycle = 0$ // Number of processing cycles of Q
7. $G_c = Table\ with\ 2^{16}\ entries,\ each\ entry\ being\ 8\ bits\ (one\ byte), initialized\ to\ 0xFF$ // This structure maintains the information exactly the same way as G, but only for crashed or hanged runs of P.

1) main:

```
for all $t_s \in$ seed set $S$ do
    reinitialize $Shm$ to all zeroes
    $Shm = \text{execute}(P, t_s)$ // Execute P with $t_s$ as input
    roundOff($Shm$)
    addToQueue($t_s, Shm$) // Add test case to the queue
end for
```

AFL-fuzz – main (continued)

```
currQ = head of $Q$
repeat
    $Q_{pr} = \text{prioritize}(Q)$                      // Prioritize test cases
    $t = \text{chooseNext}(Q, Q_{pr})$                   // Select a test case from queue
    $Q_m = \emptyset$                                   // List of new mutants
    if $t$ is not fuzzed so far then
        $Q_m$ += mutateDeterministically($t$)      // Add deterministically fuzzed ones
    end if
    $N = \text{assignFuzzingEnergy}(t)$          // Determine number of non-deterministic
                                             // mutants to produce
    $Q_m$ += mutateNonDet ($t, N$)              // Mutate $t$ non-deterministically $N$ times
    for all $t' \in Q_m$ do
        reinitialize $Shm$ to all zeroes
        $Shm = \text{execute}(P, t')$
        if ($P$ crashes or hangs) then  // Is $t'$ causing a crash or a hang of $P$?
          if isUniqueCrashOrHang($t', Shm$) then
              add $t'$ to $Q_x$             // Store it as crashing input
          end if
        else if isInterestingTestCase($t', Shm$) then // Is $t'$ of interest?
              addToQueue($t', Shm$)  // If yes, add it to queue
       end if
    end for
until user stops fuzzing                  // run this until user stops it
```

AFL-fuzz – addToQueue

2) addToQueue(t',Shm):

```
Create metadata $m$ for $t'$        // metadata $m$: size of $t'$, exec_time of $P$ with $t'$, cycle and depth at which $t'$ is discovered, and number of $bps$ covered by $t'$
append $(t', m)$ to $end\ of\ Q$
updateCoverage($G, Shm, t'$) // update G and R using t’
$for$ $i = 1$ to $sizeOf(G)$ $do$
    $if$ $Shm[i] \neq 0$ $then$     // If test input $t'$ visited branch pair $i$
      $if$ $R[i] \neq null$ $and$ score($R[i]$) $\geq$ score($t'$) $then$ // For any test input $t$,score(t) = size(t)*exec_time(t)
          $R[i] = t'$
      $end$ $if$
    $end$ $if$        // $Shm[i] \neq 0$
$end$ $for$
```

3) updateCoverage(G, Shm, t'):

```
$for$ $i = 1$ to $sizeOf(G)$ $do$
    $if$ $Shm[i] \neq 0$ $and$ $G[i] == 255$ $then$
        $R[i] = t'$
    $end$ $if$
    $if$ $Shm[i]$ & $G[i] > 0$ $then$ //Does Shm[i] have a new visit count never seen before by other test inputs?
        $G[i]$  $= G[i]$ & $\sim Shm[i]$  // zero the corresponding visited bit in G
    $end$ $if$
$end$ $for$
```

AFL-fuzz - prioritize

4) prioritize(Q):

```
$Q_{pr} = \emptyset$
$\boldsymbol{for}\ t \in Q\ \boldsymbol{do}$
    $\boldsymbol{if}\ \exists i. R[i] = t\ and\ t \notin Q_{pr}\ \boldsymbol{then}$
        add $t$ to $Q_{pr}$
    $\boldsymbol{end\ if}$
$\boldsymbol{end\ for}$
$return\ Q_{pr}$
```

AFL-fuzz - chooseNext

5) chooseNext(Q, Q_{pr}):

```
 while(true)
     Advance currQ to next element of $Q$ (to head of $Q$ in case currQ already at last element)
     if currQ wrapped around in previous step, then $cycle = cycle + 1$
     Let $t$ be the test case in $Q$ at currQ
     Let $b = \boldsymbol{random}(\mathbf{100})$        // Pick a random number between 0 - 99
     if $b < 95$ then
         if $t \in Q_{pr}$ or $t$ is not fuzzed so far then
             return $t$
       end if
     else
       return $t$
     end if
 end while
```

AFL-fuzz - assignFuzzingEnegry

```
6) assignFuzzingEnergy($t$):
    Let $N = 100.$
    Let $N_1$ be the $N$ × a factor inversely proportional to $t's$ $execution\ time$.
    Let $N_2$ be $N_1$ × a factor based on $number\ of\ branch\ pairs$ covered by $t$.
    Let $N_3$ be $N_2$ × a factor based on $cycle$ of $t'$s discovery.
    Let $N_4$ be $N_3$ × a factor based on $depth$ of $t'$s discovery.
  $return\ \ N_4$
```

AFL-fuzz – isInterestingTestCase

7) isInterestingTestCase(t, Shm):

```
roundOff($Shm$)    // rounds of every element in Shm to a power of 2
for $i = 0$ to $SizeOf(G)$ do
    if $G[i] == 255$ and $Shm[i] \neq 0$ or
        ($G[i] \neq 255$ and $Shm[i] \,\&\, G[i] > 0$) then // Is there any new visit bit in Shm[i] relative to G[i]?
        return true
    end if
end for
return false
```

8) roundOff (Shm):

```
for $i = 0$ to $SizeOf(Shm)$ do
    if $Shm[i] == 0$ then $Shm[i] = 0$                                  // all bits set to zero
    else if $Shm[i] == 1$ then $Shm[i] = 1$                             // 0000 0001
    else if $Shm[i] == 2$ then $Shm[i] = 2$                             // 0000 0010
    else if $Shm[i] == 3$ then $Shm[i] = 4$                             // 0000 0100
    else if $Shm[i] \geq 4$ and $Shm[i] \leq 7$ then $Shm[i] = 8$       // 0000 1000
    else if $Shm[i] \geq 8$ and $Shm[i] \leq 15$ then $Shm[i] = 16$     // 0001 0000
    else if $Shm[i] \geq 16$ and $Shm[i] \leq 31$ then $Shm[i] = 32$    // 0010 0000
    else if $Shm[i] \geq 32$ and $Shm[i] \leq 127$ then $Shm[i] = 64$   // 0100 0000
    else $Shm[i] = 128$                                                 // 1000 0000
end for
```

AFL-fuzz – isUniqueCrashOrHang

```
9)  isUniqueCrashOrHang($t, Shm$):
     uniqueCrashOrHangFlag = $false$
     roundOff($Shm$)    // rounds of every element in Shm to a power of 2
    $\boldsymbol{for}$  $i = 0$ $to$ $\boldsymbol{SizeOf}(G_c)$ $\boldsymbol{do}$
       $\boldsymbol{if}$ $(G_c[i] == 255$ $and$ $Shm[i] \neq 0)$ $or$
           $(G_c[i] \neq 255$ $and$ $Shm[i]\,\&\,G_c[i] > 0)$ $\boldsymbol{then}$ // Is there any new visit bit in Shm[i] relative to $G_c$[i]?
            $G_c[i] = G_c[i]\,\&\,{\sim}Shm[i]$            // zero the corresponding visited bit in $G_c$
           uniqueCrashOrHangFlag = $true$
        $\boldsymbol{end\ if}$
    $\boldsymbol{end\ for}$
    $return$ uniqueCrashOrHangFlag
```