Perspectives on search strategies in automated test input generation

Yang CAO, Yanyan JIANG, Chang XU, Jun MA, Xiaoxing MA

Frontiers of Computer Science, DOI: 10.1007/s11704-019-8281-3

Problems & Ideas

- Problems of Automated Test Input Generation
  - How do existing fuzzing and DSE techniques model the input space and manage the search procedure?
  - What portions of code are difficult to reach in the search space for the existing techniques?
- Ideas: View Test Input Generation as a Search Problem
  - An automated test input generation technique can be characterized by a three-element tuple: $\langle N, S, \mathrm{H}_0 \rangle$
  - An automated test input generation technique is an iterative procedure: $H_{i+1} \leftarrow H_i \cup S(N(H_i))$

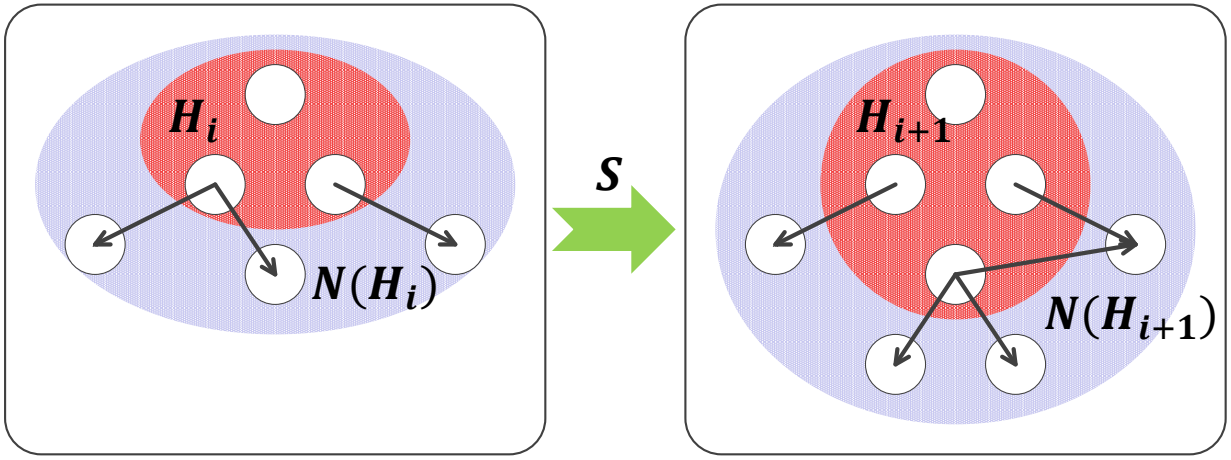

Fig. 1 Illustration of $H_{i+1} \leftarrow H_i \cup S(N(H_i))$

Main Contributions

- We proposed a unified perspective $\langle N, S, \mathrm{H}_0 \rangle$ to model and characterize the test input generation problem and its solutions, and conducted a mini survey under the unified perspective.

- We conducted an empirical study to investigate the distinctive power and limitations of fuzzing and DSE when applied to real-world programs.

Table 1 The statement coverage performance of fuzzing and DSE on the GNU CoreUtils in 1 hour

	Statement Coverage (%)		
Program	**AFLfast (a)**	**AFLfast (–help)**	**KLEE**
basename	76.9 (-23.1)	*100	*100
date	77.2 (-11.7)	*88.9	88.0 (-0.9)
dirname	72.7 (-27.3)	*100	*100
echo	81.9 (-14.9)	*96.8	78.9 (-17.9)
expr	68.1 (-16.6)	76.8 (-7.8)	*84.7
factor	91.7 (-1.3)	*93.0	33.7 (-59.4)
hostid	81.8 (-18.2)	*100	*100
logname	70.8 (-20.8)	*91.7	*91.7
nproc	68.8 (-28.6)	95.0 (-2.3)	*97.3
numfmt	51.5 (-14.5)	63.0 (-2.9)	*65.9
pathchk	78.2 (-2.7)	*80.9	72.0 (-8.9)
printenv	97.4 (-2.6)	*100	*100
printf	83.3 (-8.5)	91.6 (-0.2)	*91.8
pwd	24.2 (-9.4)	32.0 (-1.5)	*33.6
seq	72.6 (-13.2)	*85.8	78.2 (-7.6)
tr	51.6 (-15.4)	*67.0	60.5 (-6.5)
uname	82.1 (-9.3)	*91.4	90.9 (-0.5)
who	82.2 (-1.3)	*83.5	79.2 (-4.3)
whoami	77.8 (-18.5)	*96.3	*96.3
Average	**73.20**	***85.99**	**81.18**
Median	**77.23**	***91.62**	**87.95**

Table 2 The statement coverage performance of fuzzing, DSE, and hybrid on the GNU CoreUtils in 1 hour

	Statement Coverage (%)		
Program	**AFLfast**	**KLEE**	**Hybrid**
basename	*100.0	*100.0	*100.0
date	*88.9	88.0	79.8 (-9.1%)
dirname	*100.0	*100.0	*100.0
echo	96.8	78.9	*98.9 (+2.1%)
expr	76.8	84.7	*90.0 (+5.3%)
factor	*93.0	33.7	92.6 (-0.4%)
hostid	*100.0	*100.0	*100.0
logname	*91.7	*91.7	*91.7
nproc	95.0	97.3	*100.0 (+2.7%)
numfmt	63.0	65.9	*75.6 (+9.6%)
pathchk	80.9	72.0	*84.6 (+3.6%)
printenv	*100.0	*100.0	*100.0
printf	91.6	91.8	*93.7 (+1.9%)
pwd	32.0	*33.6	31.5 (-2.1%)
seq	85.8	78.2	*87.7 (+1.9%)
tr	67.0	60.5	*72.9 (+6.0%)
uname	91.4	90.9	*91.7 (+0.2%)
who	83.5	79.2	*84.4 (+0.9%)
whoami	*96.3	*96.3	*96.3
Average	**85.99**	**81.18**	***87.96**
Median	**91.62**	**87.95**	***91.67**

- In the empirical study, we also summarized the causes of 142 non-covered cases into three categories:

  [D] Dead code.

  [M] Insufficient modeling of system states, data types, or APIs.

  [A] Algorithmic limitation of the concerned $\langle N, S, \mathrm{H}_0 \rangle$.

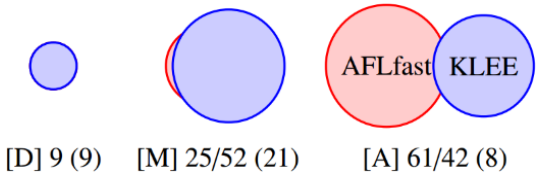

Fig. 2 Statistics of the 142 non-covered cases

- Given the mini survey and the empirical study, we finally summarized findings for future researches on automated test input generation:

  (1) Different strategies, particularly fuzzing and DSE, not only complement each other but also have common limitations.

  (2) There are challenges in exploiting the complementariness over different search strategies.

  (3) Simple code coverage may not be the proper criterion in guiding the search towards deeper states.

  (4) Program transformation fundamentally changes the search space and maybe a promising research direction.

  (5) There may be a grand unified framework to fully leverage the power of existing techniques on test input generation.