Fuzzing High-Level Synthesis Tools

Zewei Du, Yann Herklotz, Nadesh Ramanathan and John Wickerson

What is fuzzing

- Testing tools using random inputs.
- Very effective at finding bugs in compilers, where a structured input can be constructed.
- Language features can be combined in unexpected ways, which are legal but may be counter-intuitive.
- These can find corner cases that would not be tested otherwise.
- Even though these test cases might never be written by a person in practice, it is still important for tools to handle these correctly.

Example: Vivado HLS Miscompilation

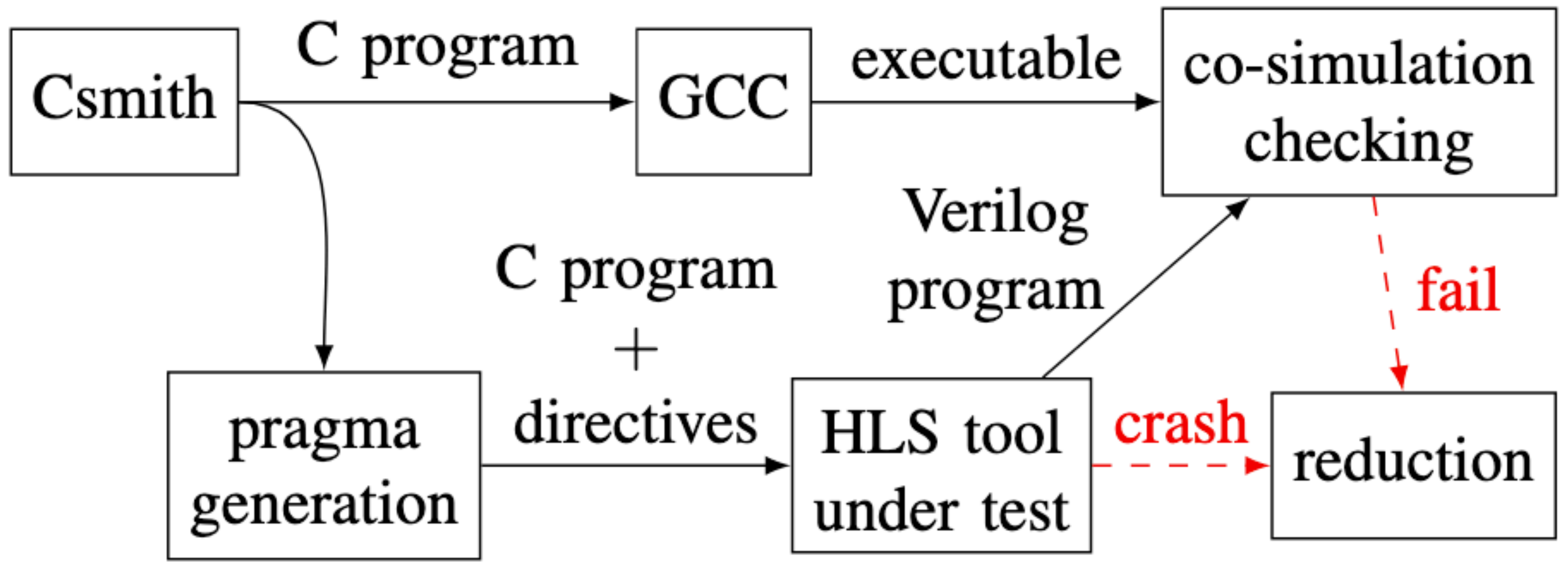
- The following code should output `0x046535FF`.
- However, the generated RTL by Vivado HLS returns `0x006535FF`.
- We initially generated a program of 113 lines which was then reduced to the following minimal example.

```
unsigned int x = 0x1194D7FF;
int arr[6] = {1, 1, 1, 1, 1, 1};

int main() {
  for (int i = 0; i < 2; i++)
    x = x >> arr[i];
  return x;
}
```

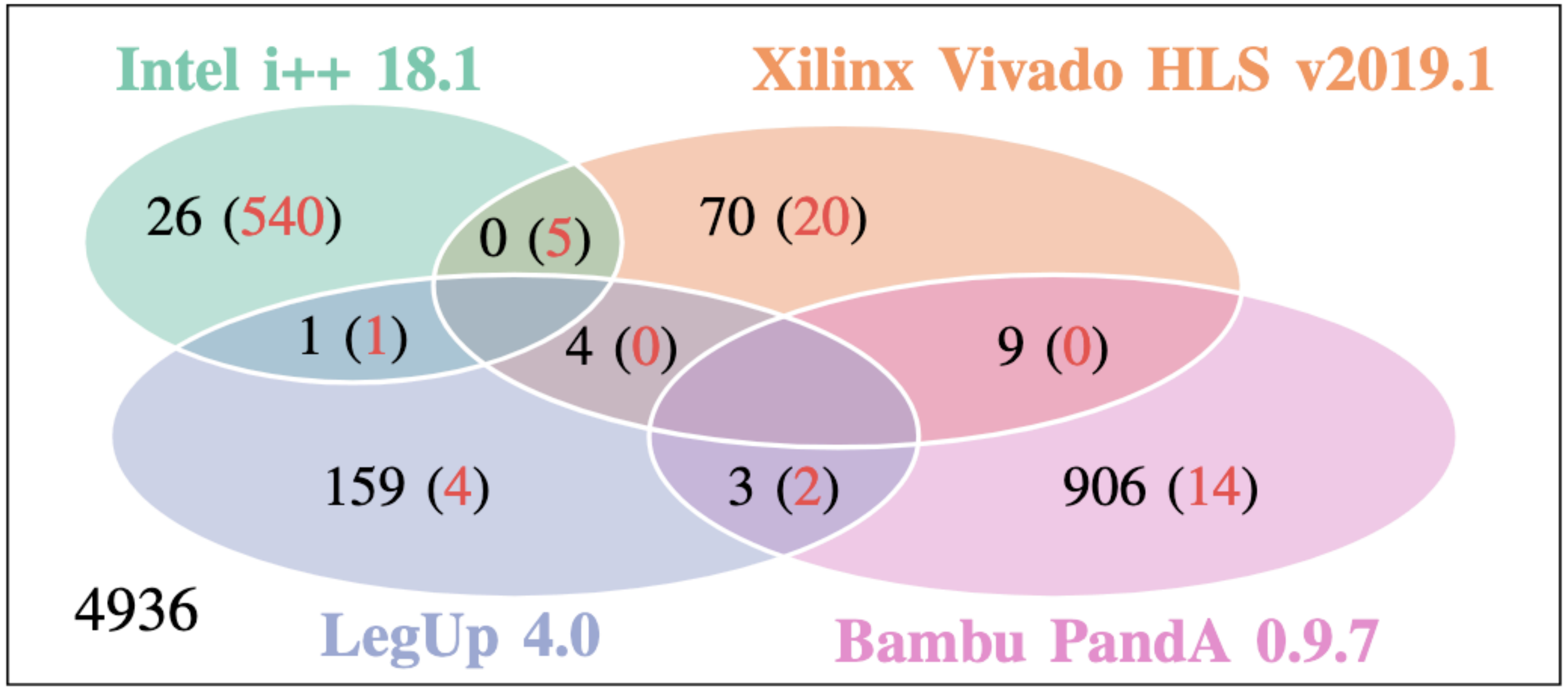
Overview of the General Workflow

- We use Csmith to generate a program C program, then augment it with HLS specific pragmas
- Pass the C program to GCC and the HLS tool under test.
- If there is a crash or a failure, the test case is automatically reduced using C-reduce.



Results for Four HLS Tools

- Some results for Intel i++, Vivado HLS, LegUp and Bambu presented as a Euler diagram.
- The red numbers stand for test cases that timed out.
- The black numbers represent failures.



External Links

- Github repository with all the test cases and failures:

  https://github.com/ymherklotz/fuzzing-hls

- Website containing a summary of all the failures that were found:

  https://ymherklotz.github.io/fuzzing-hls