Reporting Test Programs Connectivity In A Human-Readable Format

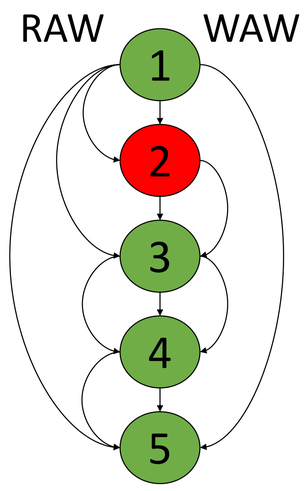
Lorenzo Cardone

DAUIN, Politecnico di Torino, Italy

This work focuses on a tool, built on top of the connectivity measure, a measure of the quality of a test program, that helps the programmer by more clearly displaying which instructions influence the result. This tool takes the program output that measures the connectivity and matches the value line by line with the source code.

Connectivity

```
subfme.  r0, r1          D: R0
                         S: R0 R1
e_add16i  r1, r0, 35     D: R1
                         S: R0 35
e_add2i.  r1, r0, 192    D: R1
                         S: R0 192
e_add2i.  r1, r1, r1     D: R1
                         S: R1 R1
subfme.  r0, r1          D: R0
                         S: R0 R1
```



Connectivity is a metric we introduced last year [1], based on the concept of information flow, designed to accompany fault coverage.

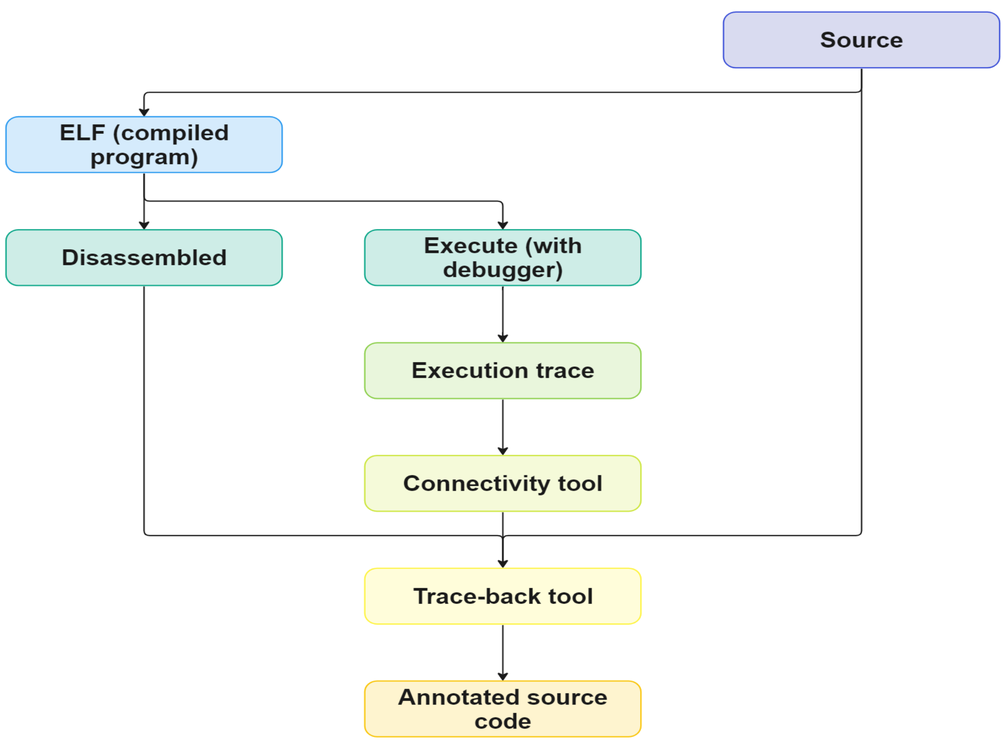
Although not as accurate, since this metric is much quicker to calculate than fault coverage, we proposed it to get quicker feedback on the quality of the test program and thus guide its writing.

Trace-back

The original connectivity software produces an output that follows the execution trace of the program, showing the list of performed instructions one after the other. This procedure makes it difficult to trace which operations are being executed in the source code, especially in the case of a conditional jump instruction or a macro.

Thanks to this software, we can read the code more naturally without being constrained by the execution flow. The output helps us navigate our projects like they were written in the first place.

Workflow



The entire flow starts from the source code. Once compiled (in ELF format for this workflow), the source code can be executed on the device to be tested. Running the compiled software with a debugger, we can extract the execution flow and calculate the connectivity based on the executed instructions. We can annotate the input program by combining the information from the disassembled ELF file, the source code, and the connectivity values.

The results are then available in text format, ready to be used by the programmer to improve the original program.

Jump code example		Trace	
li r0, 0	100%	li r0, 0	100%
cmpi r0, 0	100%	cmpi r0, 0	100%
beq jump	0%	beq jump	0%
addi r0, 1	NaN		
jump: subi r0, 1	100%	subi r0, 1	100%

Macro code example		Trace	
li r0, 0	0%	li r0, 0	0%
macro add_one r0	-		
addi r0, 1	0%	addi r0, 1	0%
end macro add_one	-		
li r0, 2	100%	li r0, 2	100%

[1] F. Angione, P. Bernardi, A. Calabrese, L. Cardone, A. Niccoletti, D. Piumatti, S. Quer, D. Appello, V. Tancorre, and R. Ugioli, “An innovative strategy to quickly grade functional test programs,” in 2022 IEEE International Test Conference (ITC), 2022, pp. 355–364.