Symbolic Execution in Selfie -a first step

Clement Poncelet, Christoph Kirsch

Symbolic Execution in Selfie - a first step -

Clément Poncelet, Christoph Kirsch
cponcelet@cs.uni-salzburg.at, ck@cs.uni-salzburg.at

Introduction

Our project consists in designing and implementing a low-level symbolic emulator following **selfie**'s philosophy: simplicity. More precisely, we tackle the question of how to implement a fully, sound and complete symbolic emulator capable of deducing properties for any machine code sequence given in input. Our first goal being to have no loss of information for at least the **selfie** code and define clearly the subset of **RISC-U** language it imply. This particular challenge is made possible thank to the simple languages handled by this independent software, alleging and preventing us from the usual assumptions in numerical analysis. All in all, we aim at using selfie's self-property to symbolically verify the selfie compilation and emulation.

Languages

High-level: **C*** **RISC-U**: Low-level

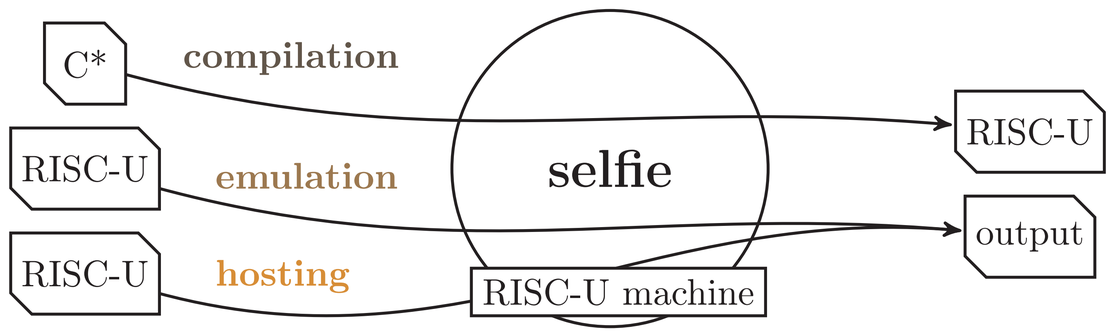
C* a tiny subset of C, supports only two data types **uint64_t** and **uint64_t***.

RISC-U a tiny subset of 64-bit RISC-V, contains **14 instructions** for **unsigned arithmetic only**.

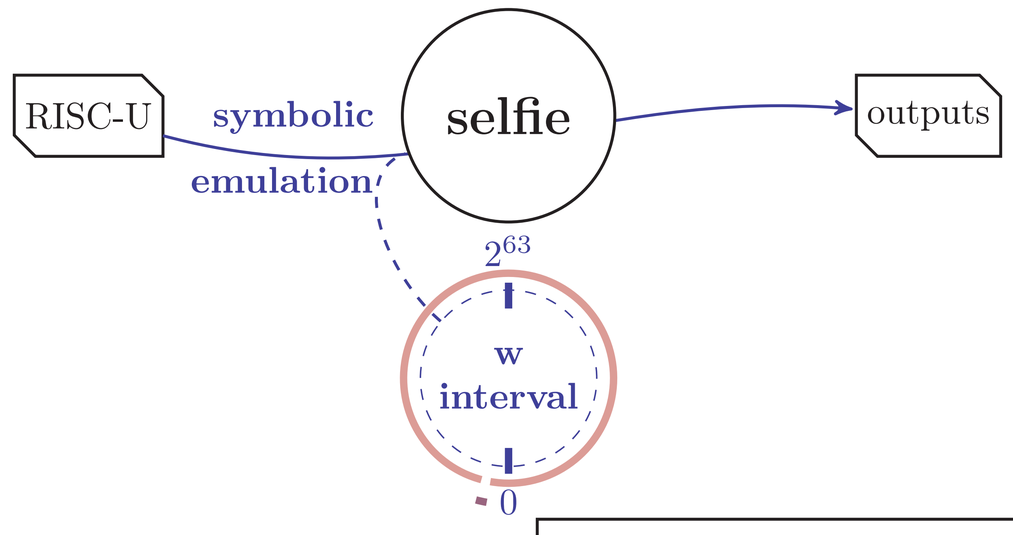
Selfie

Selfie[a] is an independent 64-bit implementation of (1) a self-**compiling** compiler written in **C*** targeting **RISC-U**, (2) a self-**executing** RISC-U emulator, and (3) a self-**hosting** hypervisor that virtualizes the emulated RISC-U machine. Selfie is implemented in a single 8k-line file and can compile, execute, and virtualize itself any number of times. Selfie has originally been developed for educational purposes but has recently become a research platform as well.

[a] `http://selfie.cs.uni-salzburg.at`



self-hosting: ./selfie **-c selfie.c -m 4 -c selfie.c -h 2 -c selfie.c**




```
x = read();
y = x + 2;
if(x == 10) return y;
return x;
```

onto RISC-U binaries

```
ecall                  | addi $t1 10
ld $t0 -8($sp)         | sub $t0 $t1 $t0
addi $t1 2             | addi $t1 1
add $t0 $t0 $t1        | sltu $t0 $t0 $t1
sd $t0 -16($sp)        | beq $t0 $zero #f
ld $t0 -8($sp)
```

then: x[10,10] y[12,12]

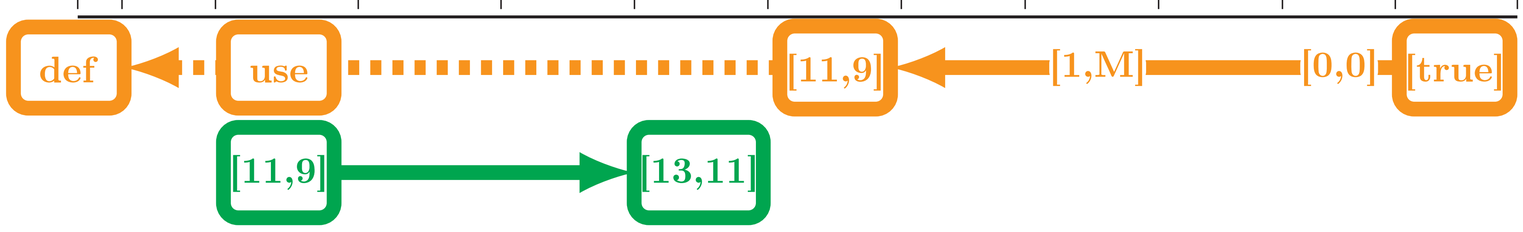
else: x[11,9] y[13,11]

Symbolic Execution

The **symbolic execution** emulates a given **RISC-U** binary, analysing numerically the instructions over **abstract** domains and operations. A current version implements **w-intervals** and initializes **read** system calls' return value as every possible value. The implementation logs each emulated instruction in a **trace** recording a triple $\langle inst, prev, value \rangle$ of the instruction binaries, the previous destination register value and the new abstract one. Moreover, each register and memory records the trace index of its last store. This data-structure allows us to apply constrains and back-propagation algorithms when the *branch on equal* instruction (**beq**) is emulated by simply moving **backward** or **forward** through the trace. An abstract domain for a complete and sound numerical analysis is still in investigation for the **selfie** binary, its simplicity allows us to avoid the usual over-approximations and fit exactly the possible values of each variable for all paths.

Logging trace

tc		...	n	n+1	n+2	n+3	n+4	n+5	n+6	n+7	n+8	**n+9**
pc		...	**ld** x	**addi**	**add**	**sd** y	**ld** x	**addi**	**sub**	**addi**	**sltu**	**beq**
prev		...	prev_{t0}	prev_{t1}	n	prev_y	n+2	n+1	n+4	n+5	n+6	
v_{start}		...	0	2	0	0	0	10	0	1	0	
v_{end}		...	MAX	2	MAX	MAX	MAX	10	MAX	1	1	



RiSE Rigorous Systems Engineering

UNIVERSITÄT SALZBURG