Formal Verification of Binary Code

Roberto Guanciale

$$\frac{\pi : \{P\}\ \alpha \rightsquigarrow \beta \cup \beta'\{Q\} \qquad \forall \alpha' \in \beta\ .\pi : \{Q\}\ \alpha' \rightsquigarrow \beta'\ \{Q\}}{\pi : \{P\}\ \alpha \rightsquigarrow \beta'\{Q\}}$$

PROSPER

seL4

Security. Performance. Proof.

P R O S P E R

seL4

Security. Performance. Proof.

PROSPER

seL4

Security. Performance. Proof.

GCC

PROSPER

seL4
Security. Performance. Proof.

```
// a0=GETBYTE(s0, 3);
ldr     r3, [r7, #84]
lsrs    r3, r3, #24
uxtb    r3, r3
str     r3, [r7, #48]
...
// v0=*(Te[0] + a0);
ldr     r3, [r7, #48]
lsls    r2, r3, #2
ldr     r3, [pc, #928] ; AesEncrypt+0x428
adds    r3, r2, r3
ldr     r3, [r3, #0]
str     r3, [r7, #32]
...
// t0 = v0  ^ v1 ^ v2 ^ v3 ^ rk[0]
```

GCC

PROSPER

seL4

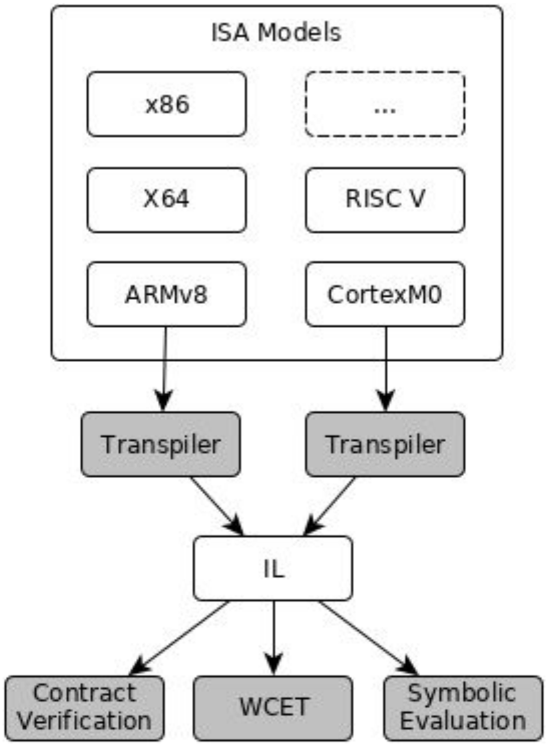
Security. Performance. Proof.

```
// a0=GETBYTE(s0, 3);
ldr      r3, [r7, #84]
lsrs     r3, r3, #24
uxtb     r3, r3
str      r3, [r7, #48]
...
// v0=*(Te[0] + a0);
ldr      r3, [r7, #48]
lsls     r2, r3, #2
ldr      r3, [pc, #928]  ; AesEncrypt+0x428
adds     r3, r2, r3
ldr      r3, [r3, #0]
str      r3, [r7, #32]
...
// t0 = v0  ^ v1 ^ v2 ^ v3 ^ rk[0]
```

GCC

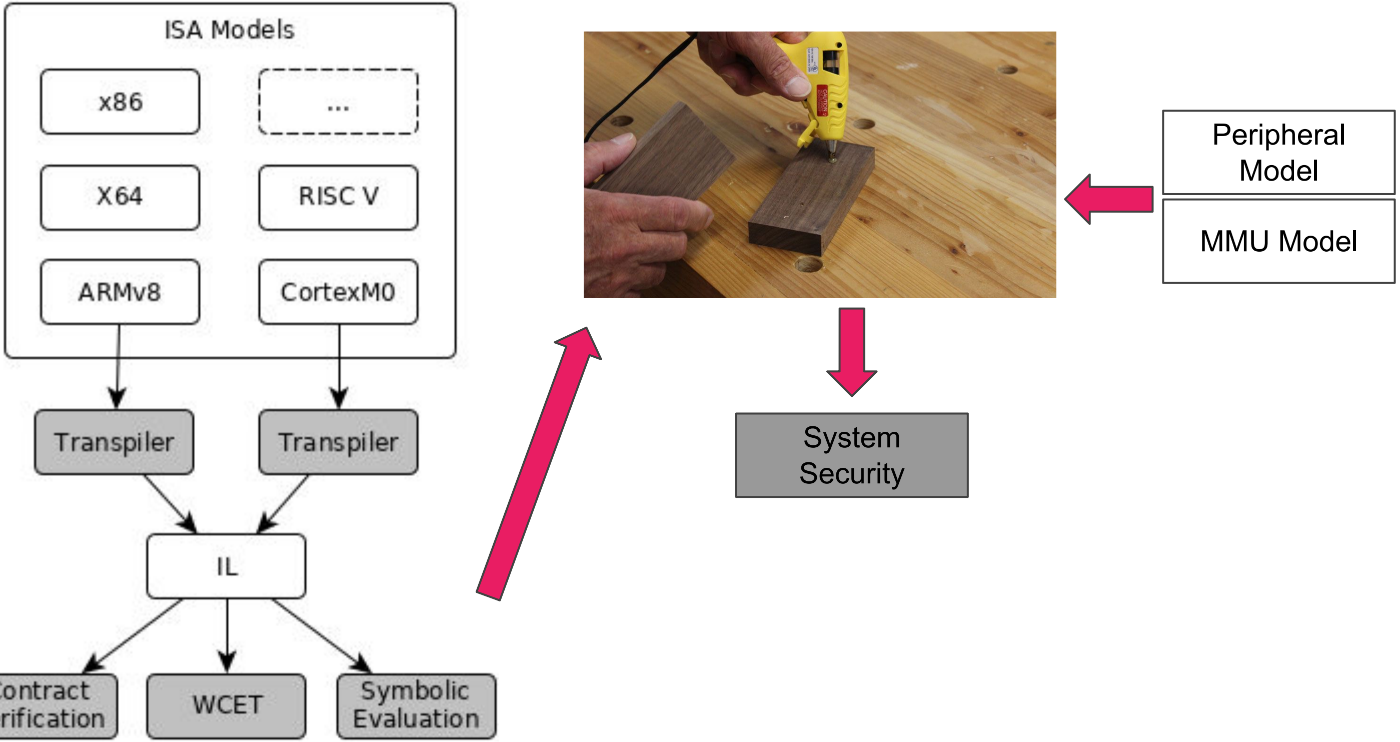
Binary Analysis Frameworks

- Valgrind
- BAP
- Angr



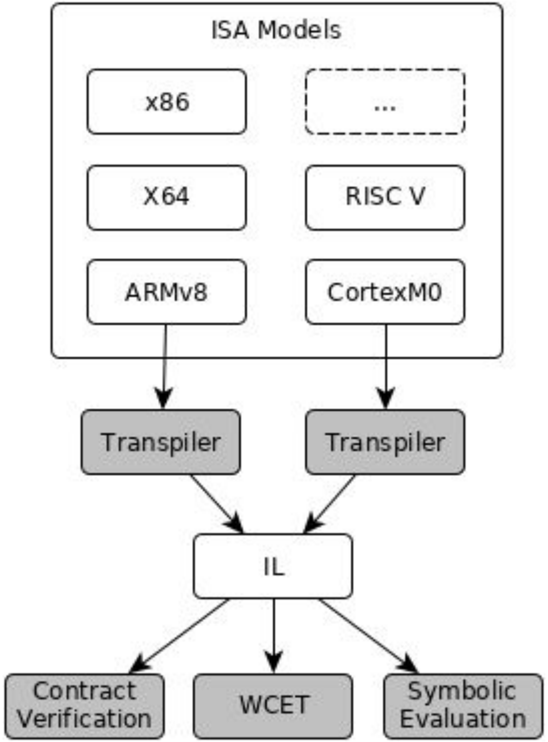
Binary Analysis Frameworks

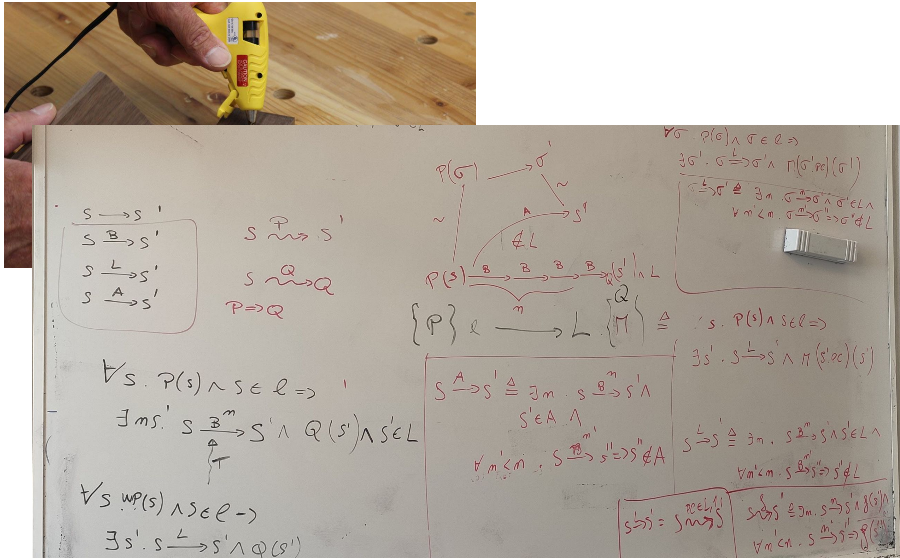
- Valgrind
- BAP
- Angr



Binary Analysis Frameworks

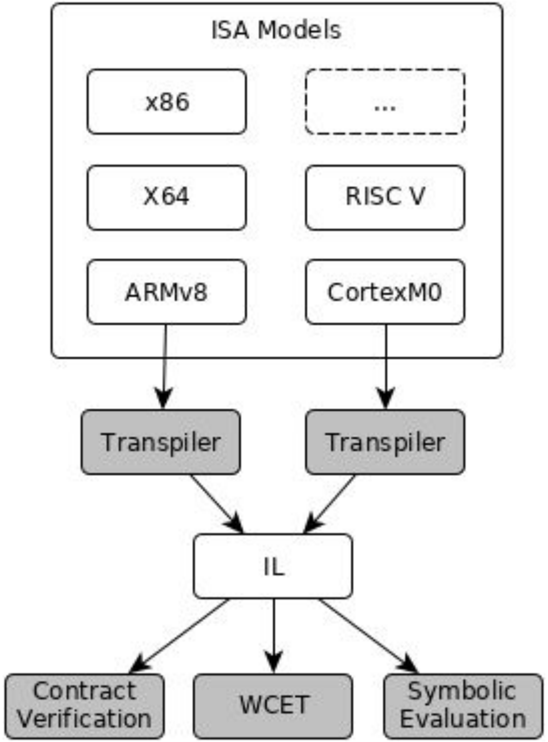
- Valgrind
- BAP
- Angr

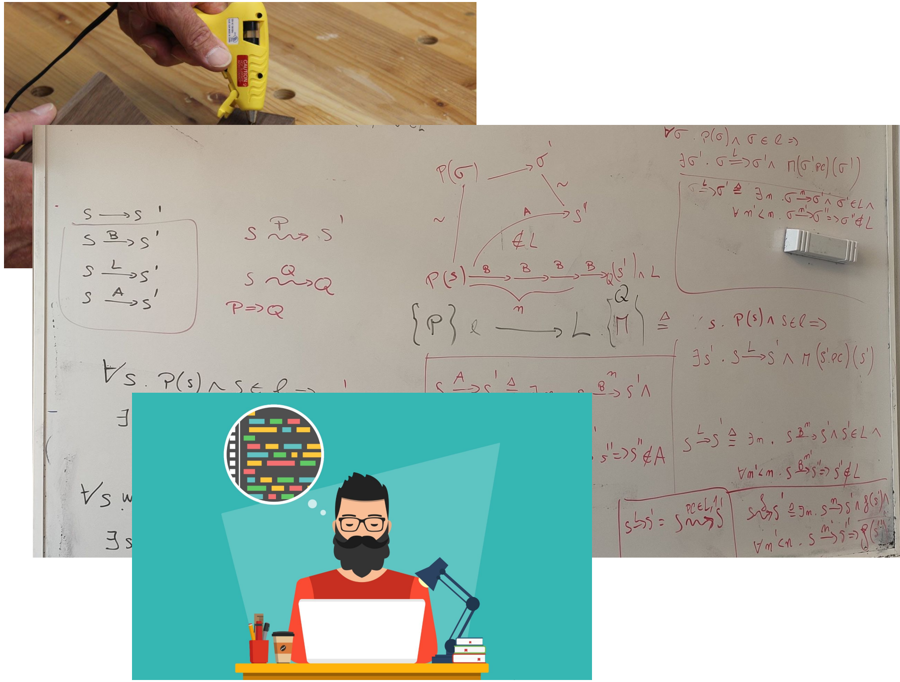




Binary Analysis Frameworks

- Valgrind
- BAP
- Angr





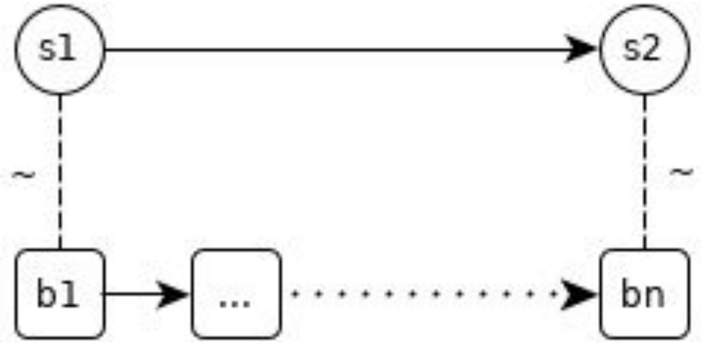
Certifying (Proof-producing) Analysis of Binaries

- Implemented using Interactive Theorem Prover (HOL4)
  - => Machine checkable proofs
- Formal semantics if ISAs (ARM/Risc-V/etc)
- Formal semantics of BinaryIntermediateRepresentation
  - Similar to LLVM IR
  - Language designed to automate analysis
    - Program not in memory / Assertions
- Verified theories and proof producing analyses
  - Transpilation
  - Contract based verification
  - ...

Certifying Transpilation

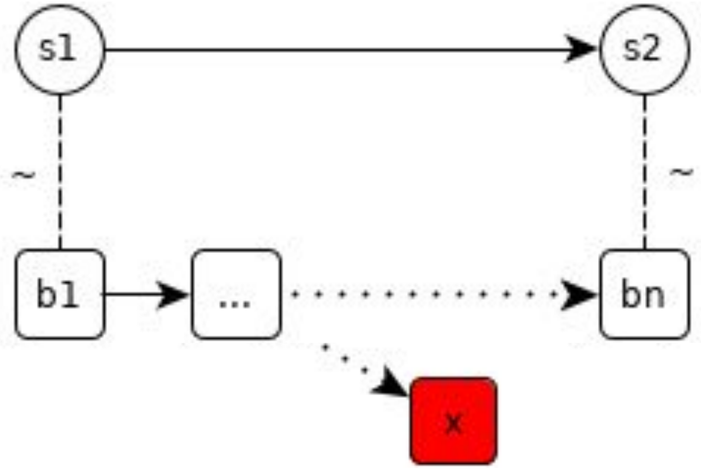
```
0: pop R1
4: push R1
```

```
[0 { R1 := MEM[SP];
     SP := SP-4;
     PC := PC+4;
     JMP 4}]
[4 { MEM := MEM with [SP<-R1];
     SP := SP+4;
     PC := PC+4;
     JMP 8}]
```



Certifying Transpilation

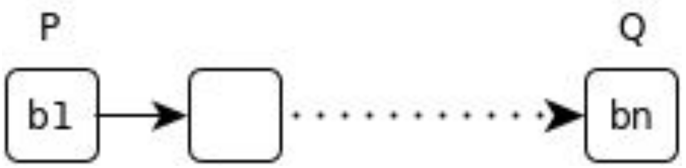
```
0: pop R1
4: push R1
```




```
[0 { R1 := MEM[SP];
     SP := SP-4;
     PC := PC+4;
     JMP 4}]
[4 { ASSERT(SP not in CODE SECTION);
     MEM := MEM with [SP<-R1];
     SP := SP+4;
     PC := PC+4;
     JMP 8}]
```

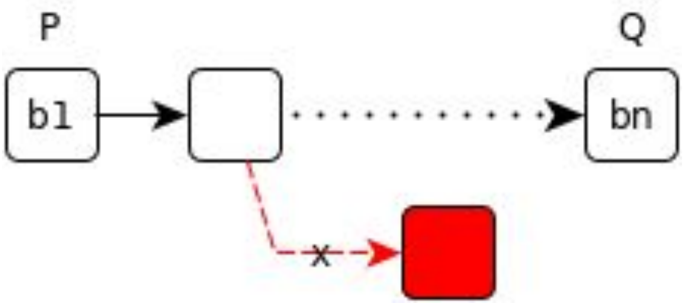
Contract Based Verification:

- For structured program
  - {P} statements {Q}
- For unstructured program?



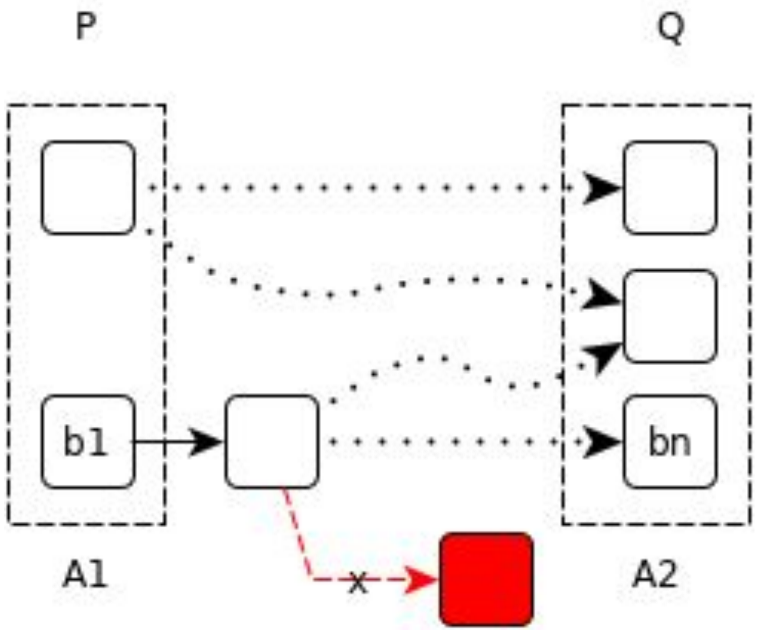
Contract Based Verification:

- For structured program
    - {P} statements {Q}
- For unstructured program?



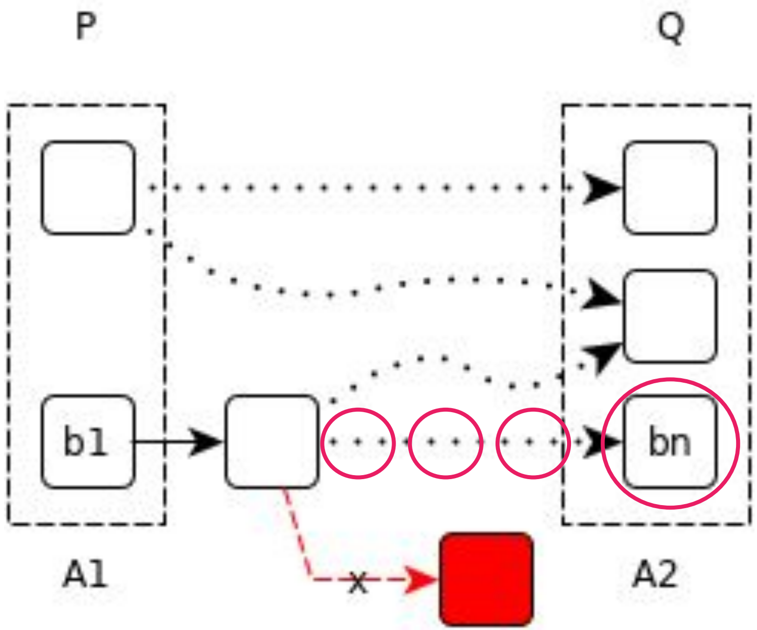
Contract Based Verification:

- For structured program
  - {P} statements {Q}
- For unstructured program?
  - {P} program: A1 -> A2 {Q}



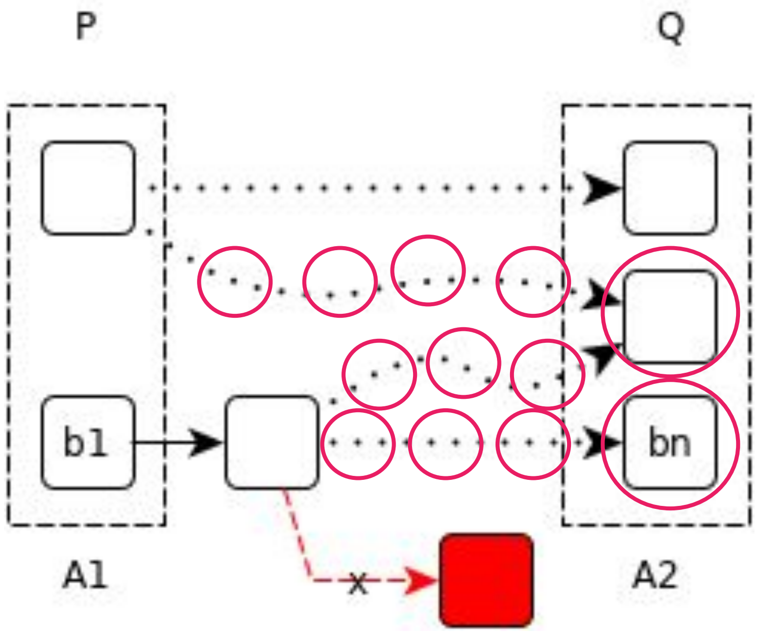
Contract Based Verification:

- For structured program
  - {P} statements {Q}
- For unstructured program?
  - {P} program: A1 -> A2 {Q}
- Semi-automatic verification
  - Weakest precondition: WP
  - SMT solver P ⇒ WP



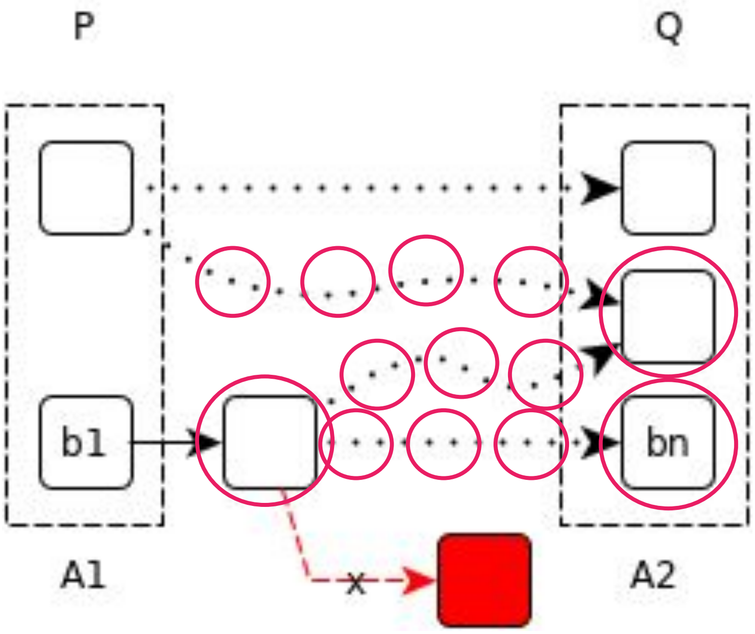
Contract Based Verification:

- For structured program
  - {P} statements {Q}
- For unstructured program?
  - {P} program: A1 -> A2 {Q}
- Semi-automatic verification
  - Weakest precondition: WP
  - SMT solver P ⇒ WP



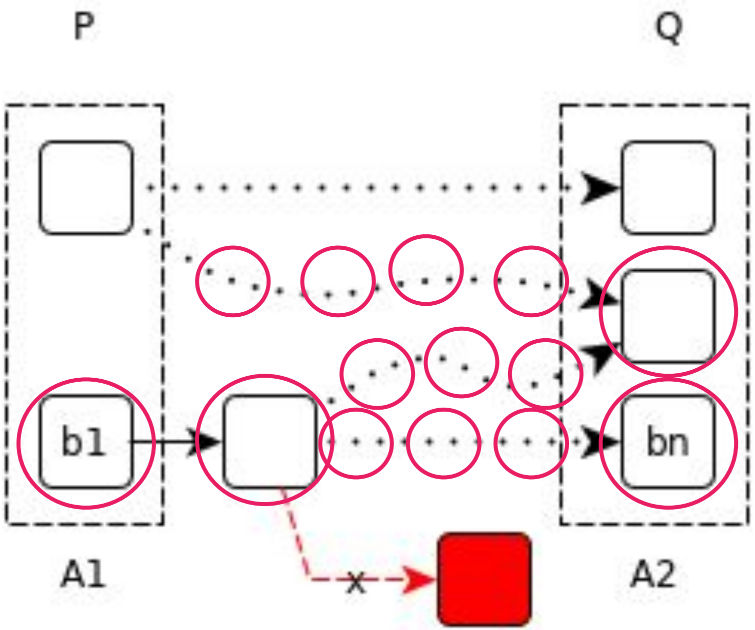
Contract Based Verification:

- For structured program
  - {P} statements {Q}
- For unstructured program?
  - {P} program: A1 -> A2 {Q}
- Semi-automatic verification
  - Weakest precondition: WP
  - SMT solver P ⇒ WP

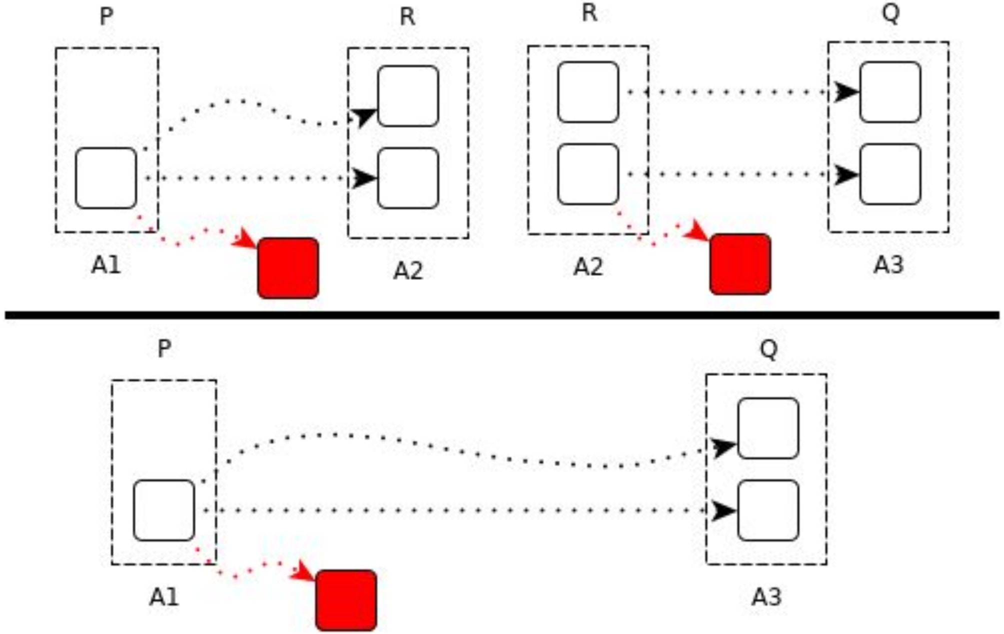


Contract Based Verification:

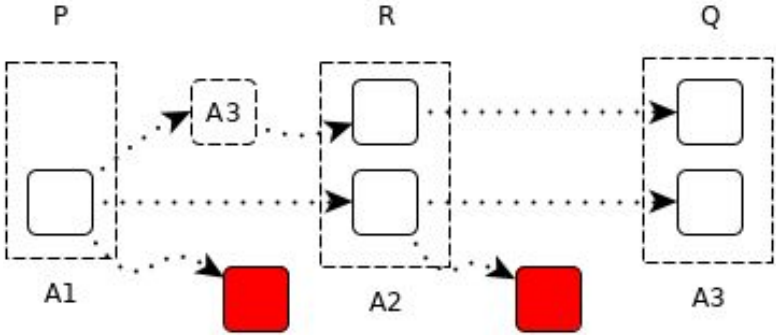
- For structured program
  - {P} statements {Q}
- For unstructured program?
  - {P} program: A1 -> A2 {Q}
- Semi-automatic verification
  - Weakest precondition: WP
  - SMT solver P ⇒ WP

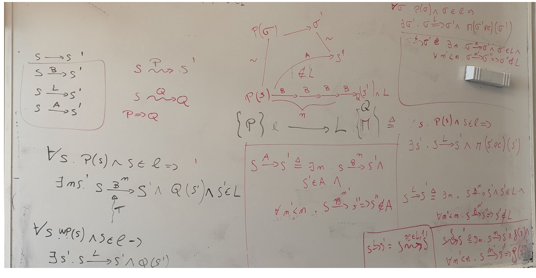


Compositional Logic For Binary Code

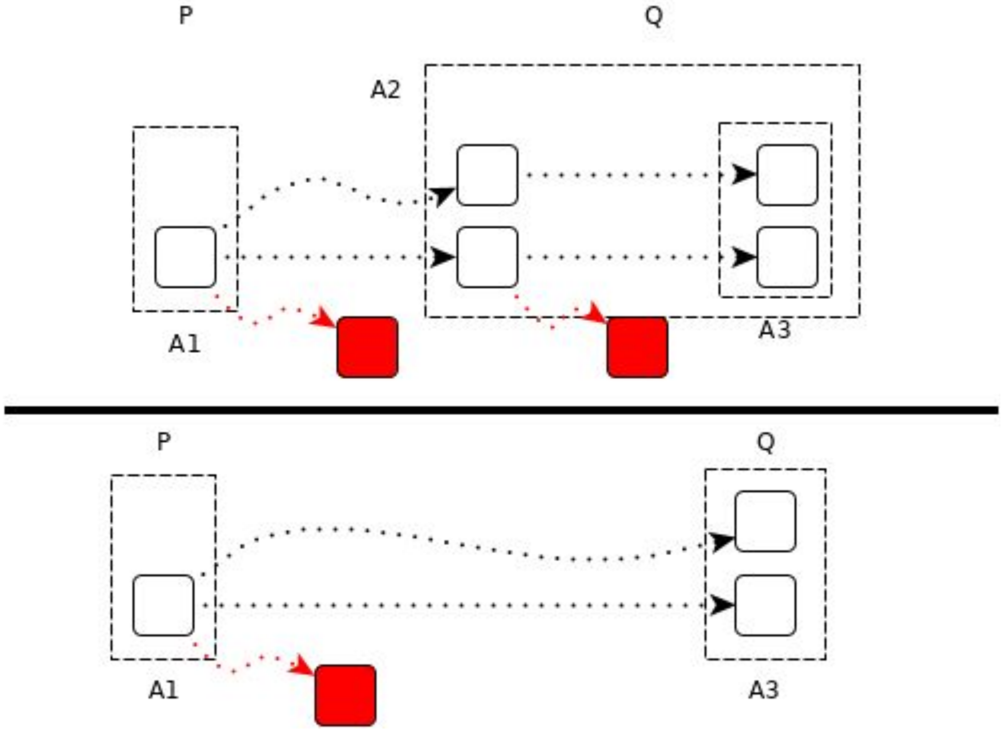


Compositional Logic For Binary Code

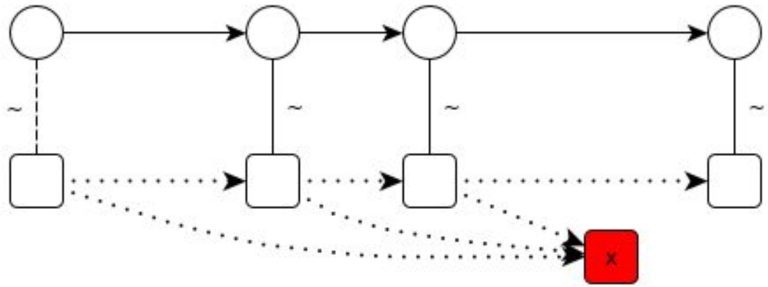




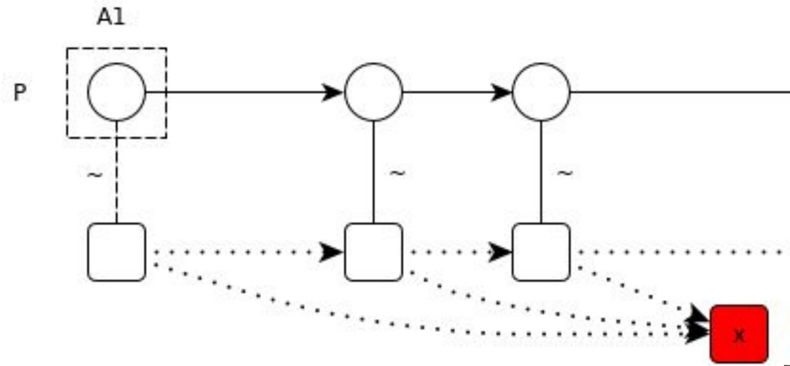
Compositional Logic For Binary Code



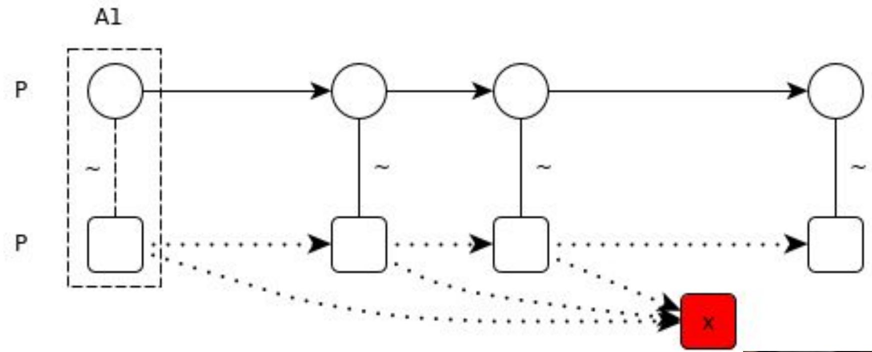
Putting things together



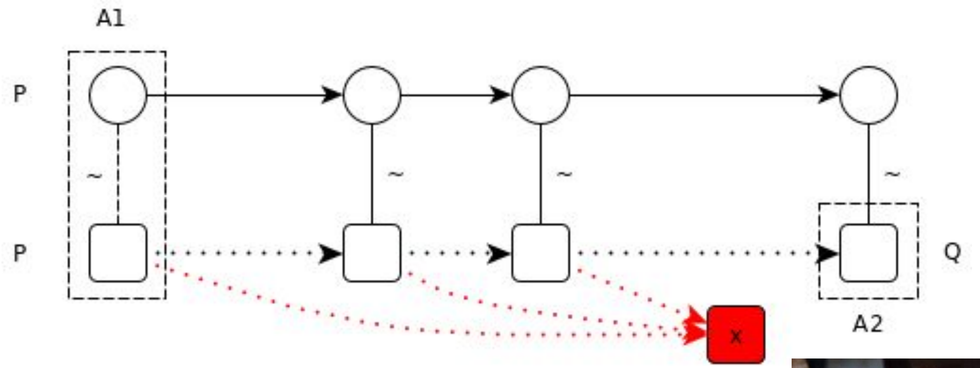
Putting things together



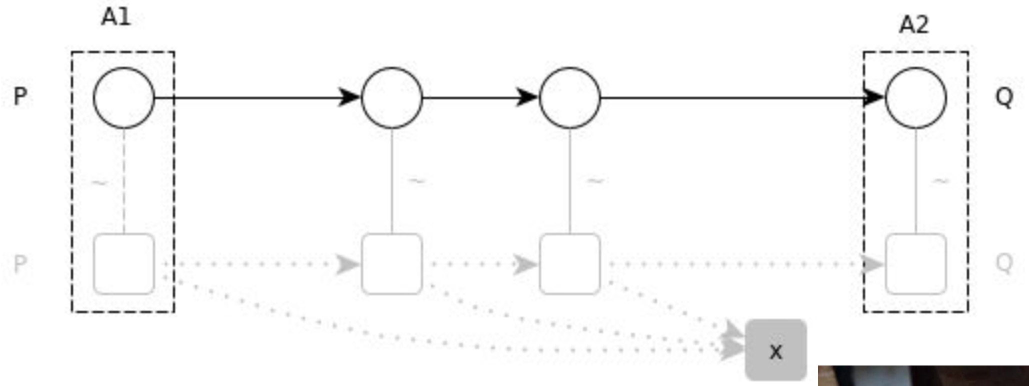
Putting things together



Putting things together



Putting things together



Real world usage

- Transpilation:
  - ~ 5 instructions / s
  - numlib / wolf-ssl / lua / SQLite / libc
  - ARMv8 / Cortex M0 / Ongoing Risc-V
- Weakest precondition
  - ~ 1 instruction / s
  - fragments consisting of 10/100 instructions (i.e. AES loop body)

Thank You

https://github.com/kth-step/HolBA

- Side channel analysis
- Symbolic execution
  - WCET
  - Translation validation
- Kernel verification