Open Universiteit
www.ou.nl

VIRGINIA TECH

Proving Program Properties

FREEK VERBEEK

SOFTWARE SECURITY

What is a proof?

A proof

- establishes a program property for all **inputs**
- establishes a program property over all **paths**
- (often) can be **mechanically** verified
- Contrast to testing:
  - establishes a program property for specific inputs and visited paths
- Contrast to static analysis:
  - often based on heuristics / tactics that skip certain program paths

“Program testing can be used to show the presence of bugs, but never to show their absence!” (Dijkstra)

Today

- How to prove a property over a program
  - crash course into Hoare logic
- How to prove properties over assembly
  - crash course into Floyd-style proving
- State-of-the-art

Hoare Triples

Partial correctness:
termination is assumed

$$\{\ P\ \}\ C\ \{\ Q\ \} \equiv \text{precondition } P \text{ ensures postcondition } Q \text{ after executing program } C$$

Examples:

$\{\ x == y\ \}$	$x := x + 3$	$\{\ x == y + 3\ \}$
$\{\ x \geq -1\ \}$	$x := 2x + 3$	$\{\ x \geq 1\ \}$
$\{\ x \geq 0\ \}$	$y := x\ \%\ 3$	$\{\ x \geq 0 \wedge y \leq 10\ \}$
$\{\ x == x' \wedge y == y'\ \}$	$t := x; x := y; y := t$	$\{\ x == y' \wedge y == x'\ \}$

Hoare Triples

```c
unsigned long pow2(unsigned exp) {
  unsigned long a = 1;
  for (unsigned i = 0; i < exp; i++) {
    a += a;
  }
  return a;
}
```

Program property:

$$\{\ \texttt{exp} == e'\ \}\ \texttt{pow2}\ \{\ \texttt{ret} == 2^{e'}\ \}$$

Hoare Logic

Substitute in P any occurrence of x with E

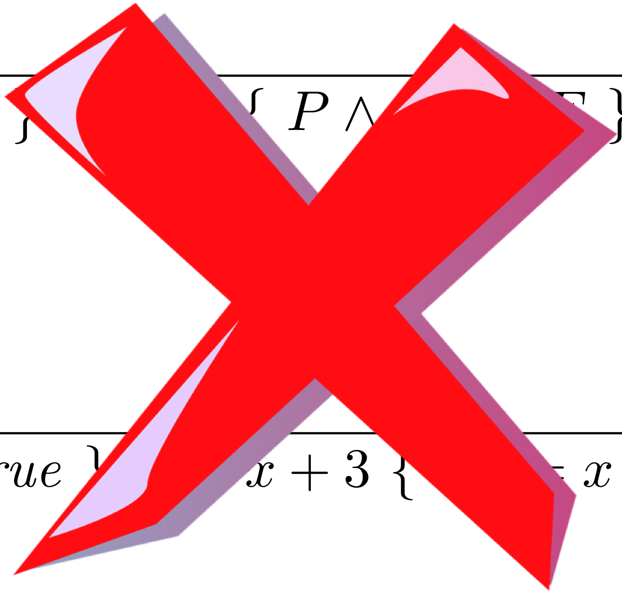
$$\frac{}{\{\ P[E/x]\ \}\ x := E\ \{\ P\ \}}\ \textsc{Assign}$$

Examples:

$$\frac{}{\{\ x+3 == y+3\ \}\ x := x+3\ \{\ x == y+3\ \}}\ \textsc{Assign}$$

$$\frac{}{\{2x+3 \geq 1\ \}\ x := 2x+3\ \{\ x \geq 1\}}\ \textsc{Assign}$$

Hoare Logic



$$\frac{}{\{\ P\ \} \qquad \{\ P \wedge \qquad \}}\ \textsc{Assign}$$

Example:

$$\frac{}{\{\ True\ \} \qquad x + 3\ \{ \qquad = x + 3\ \}}\ \textsc{Assign}$$

Hoare Logic

$$\frac{\{\ P\ \}\ C_0\ \{\ Q\ \} \qquad\qquad \{\ Q\ \}\ C_1\ \{\ R\ \}}{\{\ P\ \}\ C_0\ ;\ C_1\ \{\ R\ \}}\ \textsc{Seq}$$

Example:

$$t := x; x := y; y := t$$

$$\frac{\dfrac{}{\{\ y == y' \land x == x'\ \}\ t := x\ \{\ y == y' \land t == x'\ \}}\ \textsc{Assign} \qquad \dfrac{}{\{\ y == y' \land t == x'\ \}\ x := y\ \{\ x == y' \land t == x'\ \}}\ \textsc{Assign}}{\{\ y == y' \land x == x'\ \}\ t := x; x := y\ \{\ x == y' \land t == x'\ \}}\ \textsc{Seq}$$

Hoare Logic

$$\frac{\{\ P\ \}\ C_0\ \{\ Q\ \} \qquad \{\ Q\ \}\ C_1\ \{\ R\ \}}{\{\ P\ \}\ C_0\ ;\ C_1\ \{\ R\ \}}\ \textsc{Seq}$$

Example:

$$\frac{\dfrac{\dfrac{}{\{\ P\ \}\ t := x\ \{\ Q\ \}}\ \textsc{Assign} \qquad \dfrac{}{\{\ Q\}\ x := y\ \{\ R\ \}}\ \textsc{Assign}}{\{\ P\ \}\ t := x; x := y\ \{\ R\ \}}\ \textsc{Seq} \qquad \dfrac{}{\{\ R\ \}\ y := t\ \{\ S\ \}}\ \textsc{Assign}}{\{\ P\ \}\ t := x; x := y; y := t\ \{\ S\ \}}\ \textsc{Seq}$$

Hoare Logic

$$\frac{\{\ P\ \}\ C_0\ \{\ Q\ \} \qquad \{\ Q\ \}\ C_1\ \{\ R\ \}}{\{\ P\ \}\ C_0\ ;\ C_1\ \{\ R\ \}}\ \textsc{Seq}$$

Example:

$$t \coloneqq x; x \coloneqq y; y \coloneqq t$$

$$\dfrac{\dfrac{\dfrac{}{\{\ y == y' \land x == x'\}\ t \coloneqq x\ \{\ y == y' \land t == x'\}}\ \textsc{Assign} \qquad \dfrac{}{\{\ y == y' \land t == x'\}\ x \coloneqq y\ \{\ x == y' \land t == x'\}}\ \textsc{Assign}}{\{\ y == y' \land x == x'\ \}\ t \coloneqq x; x \coloneqq y\ \{\ x == y' \land t == x'\}}\ \textsc{Seq} \qquad \dfrac{}{\{\ x == y' \land t == x'\}\ y \coloneqq t\ \{\ x == y' \land y == x'\}}\ \textsc{Assign}}{\{\ y == y' \land x == x'\ \}\ t \coloneqq x; x \coloneqq y; y \coloneqq t\ \{\ x == y' \land y == x'\}}\ \textsc{Seq}$$

Hoare Logic

$$\frac{\{\ P \land B\ \}\ C_0\ \{\ Q\ \} \qquad \{\ P \land \neg B\ \}\ C_1\ \{\ Q\ \}}{\{\ P\ \}\ \texttt{if}\ B\ \texttt{then}\ C_0\ \texttt{else}\ C_1\ \{\ Q\ \}}\ \textsc{If}$$

$$\frac{\{\ P \land B\ \}\ C\ \{\ P\ \}}{\{\ P\ \}\ \texttt{while}\ B\ \texttt{do}\ C\ \{\ P \land \neg B\ \}}\ \textsc{While}$$

Loop invariant

$$\frac{\{\ P \land B\ \}\ C\ \{\ P\ \}}{\{\ P\ \}\ \texttt{while}\ B\ \texttt{do}\ C\ \{\ P \land \neg B\ \}}\ \textsc{While}$$

```
unsigned long pow2(unsigned exp) {
  unsigned long a = 1;
  for (unsigned i = 0; i < exp; i++) {
    a += a;
  }
  return a;
}
```

First try:

$$P \equiv i \leq e'$$

An invariant, but not strong enough.

We must find a predicate P such that:

$$\{\ P \land i < e'\ \}\ a \mathrel{+}=\ a;\ i{+}{+}\ \{\ P\ \}$$

Loop invariant

$$\frac{\{\ P \land B\ \}\ C\ \{\ P\ \}}{\{\ P\ \}\ \texttt{while}\ B\ \texttt{do}\ C\ \{\ P \land \neg B\ \}}\ \textsc{While}$$

```
unsigned long pow2(unsigned exp) {
  unsigned long a = 1;
  for (unsigned i = 0; i < exp; i++) {
    a += a;
  }
  return a;
}
```

We want to prove:

$$\{\ \texttt{exp} == e'\ \}\ \texttt{pow2}\ \{\ \texttt{ret} == 2^{e'}\ \}$$

We must find a predicate P such that:

$$\{\ P \land i < e'\ \}\ a \mathrel{+}= a;\ i{+}{+}\ \{\ P\ \}$$

$$\{\ P \land i \geq e'\ \}\ \texttt{ret} := a\ \{\ \texttt{ret} == 2^{e'}\ \}$$

Loop invariant

$$\frac{\{\ P \land B\ \}\ C\ \{\ P\ \}}{\{\ P\ \}\ \texttt{while}\ B\ \texttt{do}\ C\ \{\ P \land \neg B\ \}}\ \textsc{While}$$

```
unsigned long pow2(unsigned exp) {
  unsigned long a = 1;
  for (unsigned i = 0; i < exp; i++) {
    a += a;
  }
  return a;
}
```

Second try:

$$P \equiv a == 2^i \land i \leq e'$$

We must find a predicate P such that:

$$\{\ P \land i < e'\ \}\ a \mathrel{+}= a;\ i{+}{+}\ \{\ P\ \}$$

$$\{\ P \land i \geq e'\ \}\ \texttt{ret} := a\ \{\ \texttt{ret} == 2^{e'}\ \}$$

Loop invariant

$$a == 2^i \land i \le e' \land i < e' \implies 2a == 2^{i+1} \land i + 1 \le e'$$
$$a == 2^i \land i \le e' \land i \ge e' \implies a == 2^{e'}$$

Second try:

$$P \equiv a == 2^i \land i \le e'$$

We must find a predicate P such that:

$$\{\ P \land i < e'\ \}\ a \mathrel{+}= \ a;\ i{+}{+}\ \{\ P\ \}$$
$$\{\ P \land i \ge e'\ \}\ \texttt{ret} := a\ \{\ \texttt{ret} == 2^{e'}\ \}$$

Loop invariant

$$\frac{\{\, P \land B \,\}\ C\ \{\, P \,\}}{\{\, P \,\}\ \texttt{while}\ B\ \texttt{do}\ C\ \{\, P \land \neg B \,\}}\ \textsc{While}$$

```
unsigned long pow2(unsigned exp) {
  unsigned long a = 1;
  for (unsigned i = 0; i < exp; i++) {
    a += a;
  }
  return a;
}
```

We want to prove:

$$\{\ \texttt{exp} == e'\ \}\ \texttt{pow2}\ \{\ \texttt{ret} == 2^{e'}\ \}$$

We must find a predicate P such that:

$$\{\ P \land i < e'\ \}\ a \mathrel{+}= a;\ i{+}{+}\ \{\ P\ \}$$

$$\{\ P \land i \geq e'\ \}\ \texttt{ret} := a\ \{\ \texttt{ret} == 2^{e'}\ \}$$

$$\{\ \texttt{exp} == e'\ \}\ a := 1; i := 0\ \{\ P\ \}$$

Loop invariant

$$\frac{\{\ P \land B\ \}\ C\ \{\ P\ \}}{\{\ P\ \}\ \texttt{while}\ B\ \texttt{do}\ C\ \{\ P \land \neg B\ \}}\ \textsc{While}$$

```
unsigned long pow2(unsigned exp) {
  unsigned long a = 1;
  for (unsigned i = 0; i < exp; i++) {
    a += a;
  }
  return a;
}
```

Invariant:

$$P \equiv a == 2^i \land i \leq e'$$

We must find a predicate P such that:

$$\{\ P \land i < e'\ \}\ a \mathrel{+}= a;\ i{+}{+}\ \{\ P\ \}$$

$$\{\ P \land i \geq e'\ \}\ \texttt{ret} := a\ \{\ \texttt{ret} == 2^{e'}\ \}$$

$$\{\ \texttt{exp} == e'\ \}\ a := 1; i := 0\ \{\ P\ \}$$

Hoare Logic

```c
unsigned long pow2(unsigned exp) {
  unsigned long a = 1;
  for (unsigned i = 0; i < exp; i++) {
    a += a;
  }
  return a;
}
```

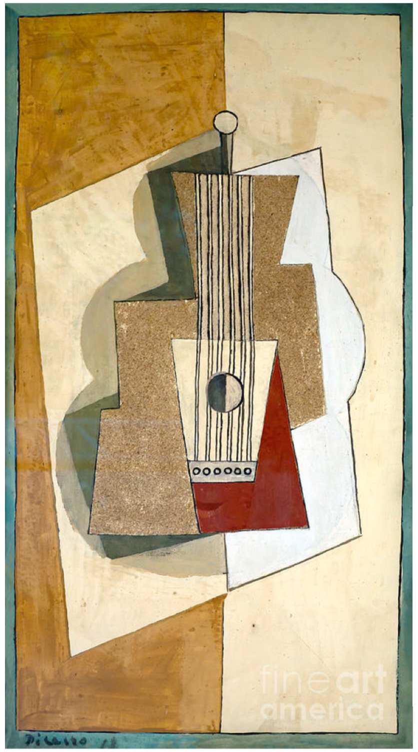
$$\dfrac{\dfrac{}{\{\ \mathtt{exp} == e'\ \}\ a := 1; i := 0\ \{\ a == 2^{i} \land i \le e'\ \}} \qquad \dfrac{\dfrac{}{\{\ a == 2^{i} \land i \le e' \land i < e'\ \}\ a \mathrel{+}= \ a;\ i{+}{+}\ \{\ a == 2^{i} \land i \le e'\ \}}}{\{\ a == 2^{i} \land i \le e'\ \}\ \mathtt{while}\ (i < e')\ \ \mathtt{do}\ a \mathrel{+}= \ a;\ i{+}{+}\ \{\ a == 2^{e'}\ \}}}{\{\ \mathtt{exp} == e'\ \}\ \ldots\ \{\ a == 2^{e'}\ \}}$$

Assembly Code

```
0.   push      rbp
1.   mov       rbp, rsp
2.   mov       dword ptr [rbp – 0x14], edi
3.   mov       qword ptr [rbp – 8], 1
4.   mov       dword ptr [rbp – 0xc], 0
5.   jmp       label_11
label_12:
6.   shl       qword ptr [rbp – 8], 1
7.   add       dword ptr [rbp – 0xc], 1
label_11:
8.   mov       eax, dword ptr [rbp – 0xc]
9.   cmp       eax, dword ptr [rbp – 0x14]
10.  jb label_12
11.  mov       rax, qword ptr [rbp — 8]
12.  pop       rbp
13.  ret
```

```
unsigned long pow2(unsigned exp) {
  unsigned long a = 1;
  for (i = 0; i < exp; i++) {
    a += a;
  }
  return a;
}
```

Assembly Code

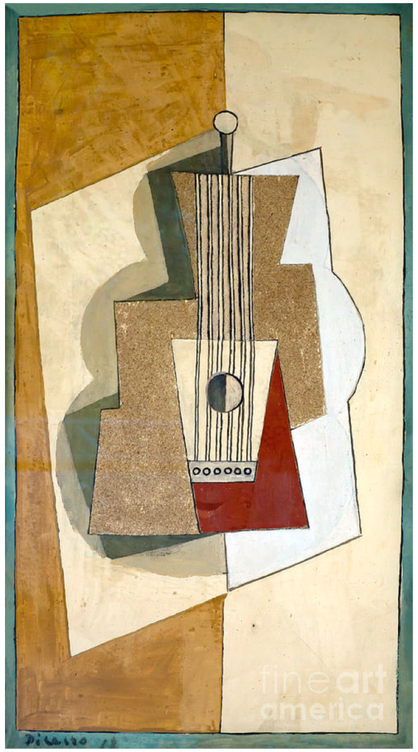


This is what we verify.

Source Code → Compiler → Binary

This is the real thing.

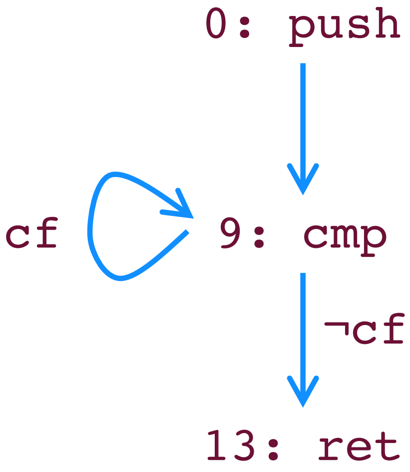
Assembly Code



Source Code	Binary
Control Flow (while, if-then-else)	Unstructured jumps (goto’s)
Variables	Unstructured memory / registers
High-level, typed operations	Bit-level, untyped operations
Datastructures	Memory Address Arithmetic

Control Flow Graph

```
0.  push      rbp
1.  mov       rbp, rsp
2.  mov       dword ptr [rbp – 0x14], edi
3.  mov       qword ptr [rbp – 8], 1
4.  mov       dword ptr [rbp – 0xc], 0
5.  jmp       label_11
label_12:
6.  shl       qword ptr [rbp – 8], 1
7.  add       dword ptr [rbp – 0xc], 1
label_11:
8.  mov       eax, dword ptr [rbp – 0xc]
9.  cmp       eax, dword ptr [rbp – 0x14]
10. jb label_12
11. mov       rax, qword ptr [rbp — 8]
12. pop       rbp
13. ret
```



Floyd Style Verification

Theorem. *Consider the CFG of a function* f*. Let each node* n *of the CFG be annotated with an invariant* P_n*. Assume that for each edge* $n_0 \xrightarrow{i_0 i_1 \ldots i_j} n_1$*, the following Hoare triple holds:*

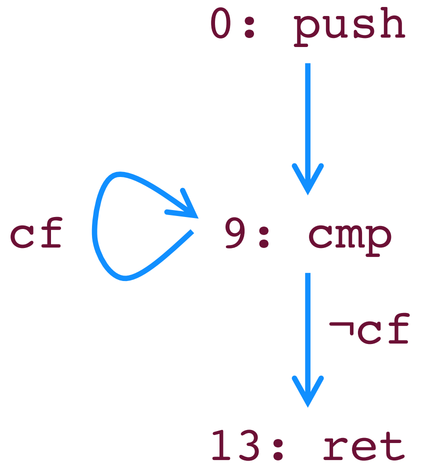
$$\{\ P_{n_0}\ \}\ i_0 i_1 \ldots i_j\ \{\ P_{n_1}\ \}$$

Let entry be the entry node and let exit be the exit node of the CFG. Then the following Hoare triple holds:

$$\{\ P_{entry}\ \}\ f\ \{\ P_{exit}\ \}$$

Control Flow Graph

$$\begin{array}{lll}
\{\ P_0\ \} & 0, 1, 2, 3, 4, 5, 9 & \{\ P_9\ \} \\
\{\ P_9 \land \texttt{cf}\ \} & 10, 6, 7, 8, 9 & \{\ P_9\ \} \\
\{\ P_9 \land \neg\texttt{cf}\ \} & 10, 11, 12, 13 & \{\ P_{13}\ \} \\
\hline
\{\ P_0\ \} & \texttt{pow2} & \{\ P_{13}\ \}
\end{array}$$



Assembly-level loop invariant

```
0.   push     rbp
1.   mov      rbp, rsp
2.   mov      dword ptr [rbp - 0x14], edi
3.   mov      qword ptr [rbp - 8], 1
4.   mov      dword ptr [rbp - 0xc], 0
5.   jmp      label_11
label_12:
6.   shl      qword ptr [rbp - 8], 1
7.   add      dword ptr [rbp - 0xc], 1
label_11:
8.   mov      eax, dword ptr [rbp - 0xc]
9.   cmp      eax, dword ptr [rbp - 0x14]
10.  jb label_12
11.  mov      rax, qword ptr [rbp – 8]
12.  pop      rbp
13.  ret
```

$$\{\ P_9 \wedge \texttt{cf}\ \}\ 10, 6, 7, 8, 9\ \{\ P_9\ \}$$

$$P_9 \equiv \begin{array}{ll} & a == 2^i \\ \wedge & i \leq e' \\ \wedge & \texttt{cf} == i < e \\ \wedge & *[\texttt{rsp}' - 16] == a \\ \wedge & *[\texttt{rsp}' - 20] == i \\ \wedge & *[\texttt{rsp}' - 28] == e' \end{array}$$

Memory (register) preservation

Under which preconditions P does a byte at address a remain the same?

$$\{\ P \land *[a] == v'\ \}\ C\ \{*[a] == v'\}$$

- Return address integrity
- Reasoning over unintended side-effects
- For each accessed memory region: is it separate from a? Overlapping? Aliasing?

Return address integrity

```
0.   push      rbp
1.   mov       rbp, rsp
2.   mov       dword ptr [rbp - 0x14], edi
3.   mov       qword ptr [rbp - 8], 1
4.   mov       dword ptr [rbp - 0xc], 0
5.   jmp       label_11
label_12:
6.   shl       qword ptr [rbp - 8], 1
7.   add       dword ptr [rbp - 0xc], 1
label_11:
8.   mov       eax, dword ptr [rbp - 0xc]
9.   cmp       eax, dword ptr [rbp - 0x14]
10.  jb label_12
11.  mov       rax, qword ptr [rbp – 8]
12.  pop       rbp
13.  ret
```

$$\{\ P_0 \wedge *[\texttt{rsp}'] == v'\ \}\ \texttt{pow2}\ \{*[\texttt{rsp}'] == v'\}$$

$$P_9 \equiv \begin{array}{rl} & a == 2^i \\ \wedge & i \leq e' \\ \wedge & \texttt{cf} == i < e \\ \wedge & *[\texttt{rsp}' - 16] == a \\ \wedge & *[\texttt{rsp}' - 20] == i \\ \wedge & *[\texttt{rsp}' - 28] == e' \\ \wedge & *[\texttt{rsp}'] == v' \end{array}$$

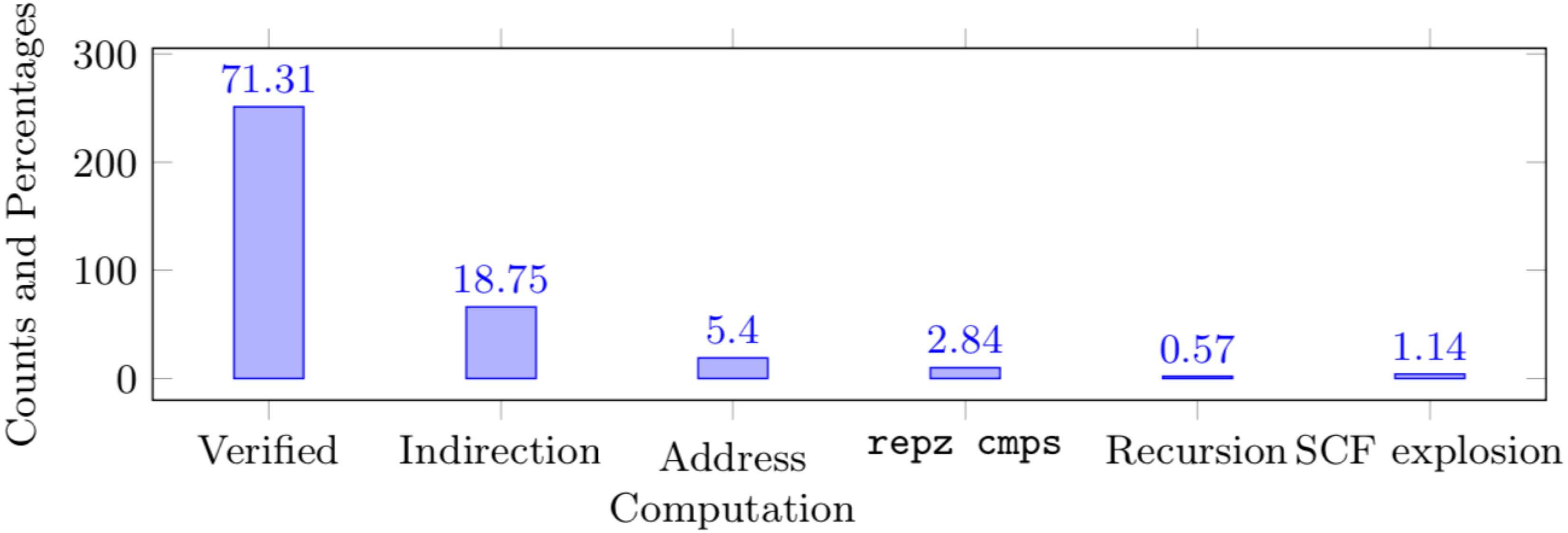
Results: Memory Preservation

Verifying the Hermitcore unikernel

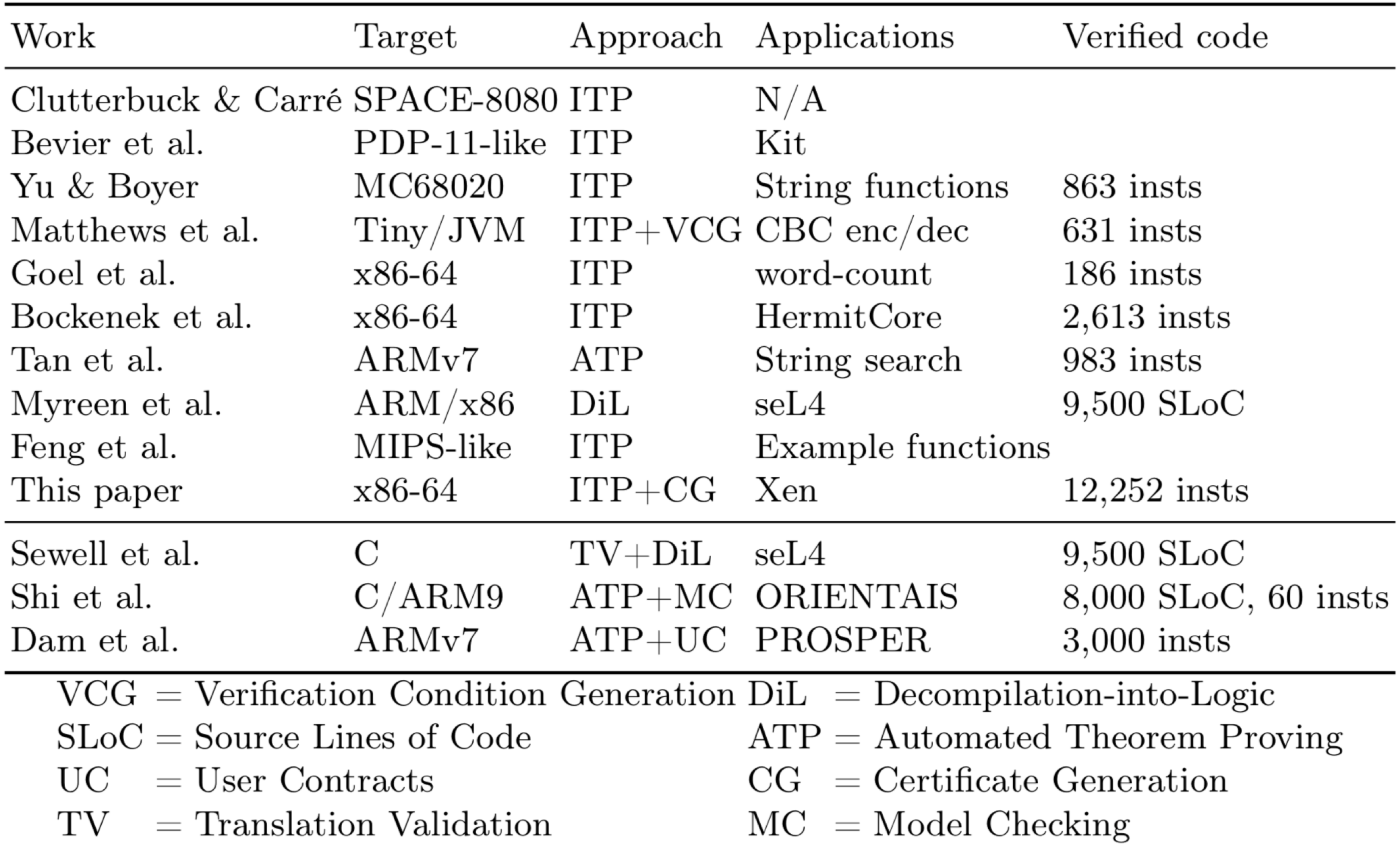
Functions	Count	SLOC (C/asm)	Loops	Recursion	Pointer args	Globals	Subcalls	-O3 done
deqeue_*	3	54/155			✓			✓
buddy_*	4	50/185	✓	✓		✓	✓	Partially
task_list_*	3	91/159			✓			✓
vring_*	3	59/97			✓			✓
string.h	6	83/358	✓		✓			
tasks.c	12	191/807	✓		✓	✓	✓	
syscall.c	11	203/593	✓		✓	✓	✓	

Results: Memory Preservation

Binaries	Function Count	Instruction Count	Loops	Manual Lines of Proof
`xenstore`	2/6	100	0	6
`xen-cpuid`	2/3	210	2	39
`qemu-img-xen`	247/343	11,942	64	1,002
Total	251/352	12,252	65	1,047



State-of-the-art



Work	Target	Approach	Applications	Verified code
Clutterbuck & Carré	SPACE-8080	ITP	N/A	
Bevier et al.	PDP-11-like	ITP	Kit	
Yu & Boyer	MC68020	ITP	String functions	863 insts
Matthews et al.	Tiny/JVM	ITP+VCG	CBC enc/dec	631 insts
Goel et al.	x86-64	ITP	word-count	186 insts
Bockenek et al.	x86-64	ITP	HermitCore	2,613 insts
Tan et al.	ARMv7	ATP	String search	983 insts
Myreen et al.	ARM/x86	DiL	seL4	9,500 SLoC
Feng et al.	MIPS-like	ITP	Example functions	
This paper	x86-64	ITP+CG	Xen	12,252 insts
Sewell et al.	C	TV+DiL	seL4	9,500 SLoC
Shi et al.	C/ARM9	ATP+MC	ORIENTAIS	8,000 SLoC, 60 insts
Dam et al.	ARMv7	ATP+UC	PROSPER	3,000 insts

VCG = Verification Condition Generation
SLoC = Source Lines of Code
UC = User Contracts
TV = Translation Validation
DiL = Decompilation-into-Logic
ATP = Automated Theorem Proving
CG = Certificate Generation
MC = Model Checking

Questions?