End-to-end Verification of Stack-space Bounds for C Programs

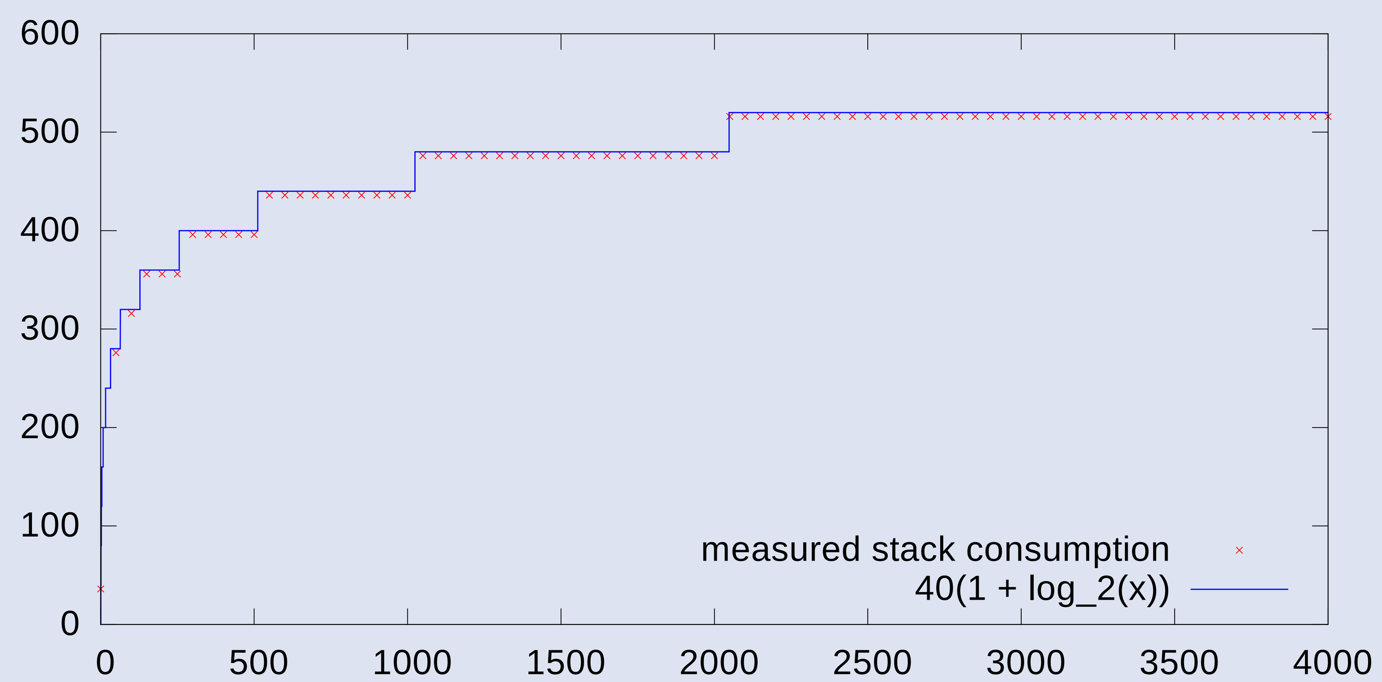
Quentin Carbonneaux, Jan Hoffmann, Tahina Ramananandro and Zhong Shao

Yale

"With experience, one learns the standard, scientific way to compute the proper size for a stack: Pick a size at random and hope."
— Jack Ganssle, *The Art of Designing Embedded Systems*

Contributions

1. Prove that CompCert preserves the stack consumption of C programs by compilation.
2. Design and prove sound a Quantitative Hoare Logic that infers stack bounds on C programs.
3. Implement an automatic procedure to derive proofs in the previous logic on simple code.



A New Quantitative CompCert

- We propose to add call and return events to CompCert traces.
- Example trace

$$call(f), ret(f), call(f), call(g), ret(g)$$

- *Event metrics* assign weights to program events.
- We define value and weight of program traces:

$$V_M(\epsilon) = 0$$
$$V_M(\nu \cdot t) = M(\nu) + V_M(t)$$
$$W_M(t) = \sup_{t_1}\{V_M(t_1) \mid t = t_1 +\!\!+ t_2\}.$$

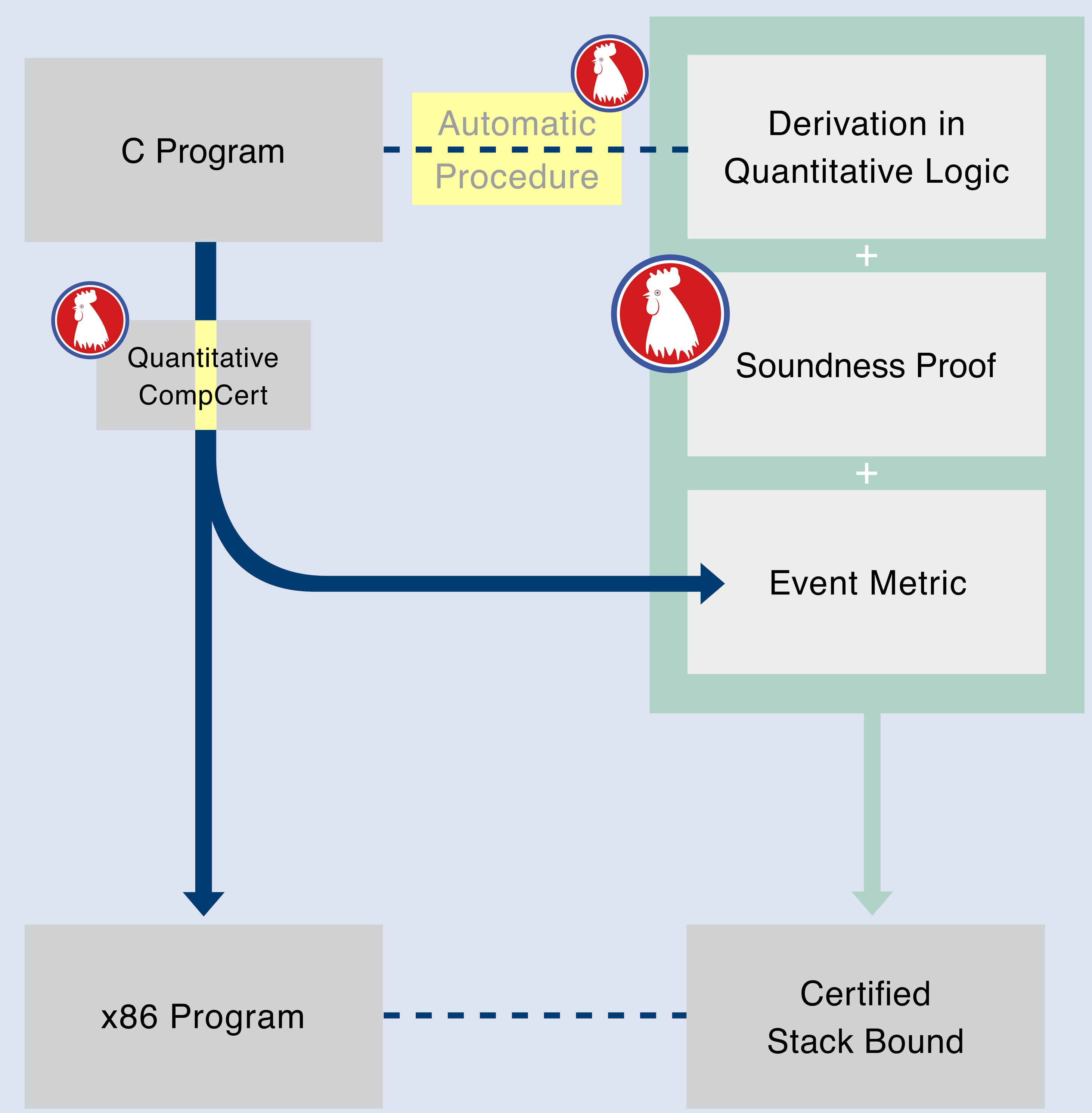
- M is the event metric, it must satisfy

$$M(call(f)) + M(ret(f)) = 0 \quad M(call(f)) > 0.$$

A New Quantitative Logic

- We define a logic inspired by Hoare logics to bound stack consumption.
- Assertions on the program state are extended to map to $\mathbb{N} \cup \{\infty\}$.
- $\bot$ is now $\_ \mapsto \infty$, $\top$ is refined by $\mathbb{N}$, $\wedge$ is $+$, $\vee$ is min, and so on.

System Overview



Hoare Logic for Quantitative Properties

$$\Gamma \vdash \{Q^s\}\ \text{skip}\ \{Q\}\ (\textsc{Skip}) \qquad \Gamma \vdash \{Q^b\}\ \text{break}\ \{Q\}\ (\textsc{Break})$$

$$\frac{P = \lambda(\theta, H)\,.\,Q^s(\theta[x \mapsto [\![E]\!]_{(\theta,H)}], H)}{\Gamma \vdash \{P\}\ x = E\ \{Q\}}\ (\textsc{AssignL})$$

$$\frac{\Gamma \vdash \{P\}\ S_1\ \{(R, Q^b, Q^r)\} \qquad \Gamma \vdash \{R\}\ S_2\ \{Q\}}{\Gamma \vdash \{P\}\ S_1; S_2\ \{Q\}}\ (\textsc{Seq})$$

$$\frac{\Gamma(f) = (P_f, Q_f) \quad P = \lambda(\theta, H)\,.\,P_f([\![E]\!]_{(\theta,H)}, H) \qquad Q = \lambda(\theta, H)\,.\,Q_f([\![x]\!]_{(\theta,H)}, H)}{\Gamma \vdash \{P + M(f)\}\ x = f(E)\ \{(Q + M(f), \bot, \bot)\}}\ (\textsc{Call})$$

$$\frac{c \geq 0 \qquad \{P\}\ S\ \{Q\}}{\{P + c\}\ S\ \{Q + c\}}\ (\textsc{Frame})$$

$$\frac{P \geq P' \qquad \{P'\}\ S\ \{Q'\} \qquad Q' \geq Q}{\{P\}\ S\ \{Q\}}\ (\textsc{ConseQ})$$

Automatically and Manually Verified Bounds

File Name / LOC	Function Name	Stack Bound
mibench/sec/blowfish.c	BF_encrypt	40 bytes
(233 LOC)	BF_options	8 bytes
	BF_ecb_encrypt	80 bytes
mibench/sec/pgp/md5.c	MD5Init	16 bytes
(335 LOC)	MD5Update	168 bytes
	MD5Transform	128 bytes
mibench/tele/fft.c	IsPowerOfTwo	16 bytes
(195 LOC)	NumberOfBitsNeeded	24 bytes
	fft_float	160 bytes
certikos/vmm.c	palloc	48 bytes
(608 LOC)	pfree	40 bytes
	mem_init	72 bytes
	pmap_init	176 bytes
	pt_init	152 bytes
certikos/proc.c	enqueue	48 bytes
(819 LOC)	dequeue	48 bytes
	sched_init	232 bytes
	thread_spawn	96 bytes

Function Name	Manually Verified Stack Bound
bsearch(x, lo, hi)	$40(1 + \log_2(\text{hi} - \text{lo}))$ bytes
fib(n)	$24(\text{n} + 1)$ bytes
qsort(a, lo, hi)	$48(\text{hi} - \text{lo} + 1)$ bytes
filter_pos(a, sz, lo, hi)	$48(\text{hi} - \text{lo} + 1)$ bytes
sum(a, lo, hi)	$32(\text{hi} - \text{lo} + 1)$ bytes
fact_sq(n)	$40 + 24\text{n}^2$ bytes

Logic Soundness

If $\vdash \{P\}s\{Q\}$ then

$$\forall \sigma\, \sigma'\, t\, M,\ (s, \sigma) \to^t (\text{skip}, \sigma') \Rightarrow W_M(t) < P(M, \sigma)$$

The implemented version is stronger and uses postconditions.

Hoare-like Reasonning for Stack Bounds

$\{Z = \log_2(\text{h}_\sigma - \text{l}_\sigma) \Rightarrow M_b \cdot Z\}$

```
bsearch(x,l,h) {
  if (h-l <= 1) return l;
```

$\{(Z > 0 \wedge Z = \log_2(\text{h}_\sigma - \text{l}_\sigma)) \Rightarrow M_b \cdot Z\}$

```
  m = (h+l)/2;
```

$\{(Z > 0 \wedge Z = \log_2(\text{h}_\sigma - \text{l}_\sigma) \wedge \text{m}_\sigma = \frac{\text{h}_\sigma + \text{l}_\sigma}{2}) \Rightarrow M_b \cdot Z\}$

```
  if (a[m]>x) h = m else l = m;
```

$\{[Z - 1 = \log_2(\text{h}_\sigma - \text{l}_\sigma) \Rightarrow M_b \cdot (Z - 1)] + M_b\}$

```
  return bsearch(x,l,h);
```

$\{[M_b \cdot (Z - 1)] + M_b\}$

```
}
```

$\{M_b \cdot Z\}$