“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*

### Contribitions

1. Prove that CompCert prescribes 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**

  \[
  \begin{array}{l}
  call(f). ret(f). call(g). ret(g) \\
  \end{array}
  \]

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

  \[
  \begin{array}{l}
  V_M(f) = 0 \\
  V_M(v \cdot i) = M(v) + V_M(f) \\
  W_M(t) = \sup\{V_M(h) \mid I = I_1 + I_2\}. \\
  \end{array}
  \]

- **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** \(N \cup \{\infty\} \) .
- **\(\perp\)** is now \(\perp \to \infty\), \(T \) is refined by \(N, \perp\) is \(+\), \(\vee\) is min, and so on.

### System Overview

- **C Program**
- **Systematic Derivation of Quantitative Properties**
- **Soundness Proof**
- **Event Metric**
- **x86 Program**
- **Certified Stack Bound**

### Hoare Logic for Quantitative Properties

\[
\Gamma \vdash \{Q^0\} \text{skip} \{Q\} (\text{SKIP}) \quad \Gamma \vdash \{Q^0\} \text{break} \{Q\} (\text{BREAK})
\]

\[
P = \lambda(\theta, H). Q^0([E]_{[\theta, H]}, H)
\]

\[
\Gamma \vdash \{P\} x = E (Q)
\]

\[
\Gamma \vdash \{S_1\} \{(R, \sigma, Q')\} (\text{SEQ})
\]

\[
\Gamma \vdash \{P\} S_1, S_2 (Q)
\]

\[
(P) f = (P_t, Q_t)
\]

\[
\Gamma \vdash \{P + M(f)\} x = f(E) \{(Q + M(f)), \perp, \perp\} (\text{CALL})
\]

\[
c \geq 0 \quad \frac{\{P\} S \{Q\}}{\{P + c\} S \{Q + c\}} (\text{FRAME})
\]

\[
P \geq P' \frac{\{P\} S \{Q'\}}{\{P\} S \{Q\}} (\text{CONSEQ})
\]

### Automatically and Manually Verified Bounds

<table>
<thead>
<tr>
<th>Function Name / LOC</th>
<th>File Name</th>
<th>Stack Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>bsearch((x, l, h))</td>
<td>BF_encrypt</td>
<td>40 bytes</td>
</tr>
<tr>
<td>MD5Update ((335 \text{ LOC}))</td>
<td>MD5Update</td>
<td>168 bytes</td>
</tr>
<tr>
<td>sec/tele/fft.c</td>
<td>MD5Transform</td>
<td>128 bytes</td>
</tr>
<tr>
<td>pfree ((40\text{ bytes LOC}))</td>
<td>pfree</td>
<td>40 bytes</td>
</tr>
<tr>
<td>mem_init</td>
<td>mem_init</td>
<td>72 bytes</td>
</tr>
<tr>
<td>pt_init</td>
<td>pt_init</td>
<td>152 bytes</td>
</tr>
<tr>
<td>enque</td>
<td>enque</td>
<td>48 bytes</td>
</tr>
<tr>
<td>dequeue</td>
<td>dequeue</td>
<td>48 bytes</td>
</tr>
<tr>
<td>sched_init</td>
<td>sched_init</td>
<td>232 bytes</td>
</tr>
<tr>
<td>thread_spawn</td>
<td>thread_spawn</td>
<td>96 bytes</td>
</tr>
</tbody>
</table>

**Logic Soundness**

If \(\vdash \{P\} s\{Q\}\) then

\[
\forall \sigma' \in M, (s, \sigma) \to (s', \sigma') \Rightarrow W(t) < P(M, \sigma)
\]

The implemented version is stronger and uses postconditions.

### Hoare-like Reasoning for Stack Bounds

\[
\begin{align*}
\{Z = \log_2(h_n - l_n) \Rightarrow M_0 \cdot Z\} \\
\text{bsearch}(x, 1, h) \{ \\
\quad \text{if } (h - 1) = 1 \text{ return } 1; \\
\quad \{Z > 0 \land Z = \log_2(h_n - l_n) \Rightarrow M_0 \cdot Z\} \\
\text{if } (a[n] \times x) = h \text{ else } l = m; \\
\quad \{Z > 0 \land Z = \log_2(h_n - l_n) \land m = h / 2 \Rightarrow M_0 \cdot Z\} \\
\text{return } \text{bsearch}(x, 1, h); \\
\quad \{M_0 \cdot (Z - 1) \Rightarrow M_0\} \\
\} \\
\{M_0 \cdot Z\}
\end{align*}
\]