End-to-end Verification of Stack-space Bounds for C Programs
Quentin Carbonneaux, Jan Hoffmann, Tahina Ramananandro and Zhong Shao

“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.
- Event metrics assign weights to program events.
- We define value and weight of program traces:
  \[ V_M(t) = 0 \]
  \[ W_M(t) = \text{sup} \{ V_M(h) \mid t = h + t \} \]
- \( M \) is the event metric, it must satisfy
  \[ M(\text{call}(t)) + M(\text{ret}(t)) = 0 \quad M(\text{call}(t)) > 0. \]

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 \( \mathbb{N} \cup \{ \infty \} \).

System Overview
- C Program
- Automatic Procedure
- Derivation in Quantitative Logic
- Soundness Proof
- Event Metric
- x86 Program
- Certified Stack Bound

Hoare Logic for Quantitative Properties
\[
\begin{align*}
\Gamma \vdash \{ Q \} \text{skip} \{ Q \} & \quad \Gamma \vdash \{ Q \} \text{break} \{ Q \} \\
\Gamma \vdash \{ P \} x = E \{ Q \} & \quad \Gamma \vdash \{ P \} S_1 \{ (R, Q, \delta') \} \quad \Gamma \vdash \{ R \} S_2 \{ Q \}
\end{align*}
\]

Logic Soundness
- If \( \vdash \{ P \} s \{ Q \} \) then
  \[ \forall \sigma' \exists \tau \quad M(s, \sigma) \rightarrow \tau (\text{skip, } \sigma') \Rightarrow W_M(t) < P(M, \sigma) \].
- The implemented version is stronger and uses postconditions.

Hoare-like Reasoning for Stack Bounds
\[
\begin{align*}
\{ Z = \log_2(h_l - l_h) \Rightarrow M_0 \cdot Z \} \\
\text{before}(x, 1, h) \{ \\
\quad \text{if } (h_{l-1} = 1) \text{ return } 1; \\
\quad \{ Z > 0 \wedge Z = \log_2(h_{l-1} - l_h) \Rightarrow M_0 \cdot Z \} \\
\quad \{ Z > 0 \Rightarrow Z = \log_2(h_{l-1} - l_h) \wedge m = \frac{h_{l-1}}{2} \Rightarrow M_0 \cdot Z \}
\end{align*}
\]

Automatically and Manually Verified Bounds

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Stack Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>mibench/sec/blowfish</td>
<td>40 bytes</td>
</tr>
<tr>
<td>mibench/sec/ebcd</td>
<td>8 bytes</td>
</tr>
<tr>
<td>mibench/sec/ebcd_encrypt</td>
<td>80 bytes</td>
</tr>
<tr>
<td>mibench/sec/pgp/md5</td>
<td>16 bytes</td>
</tr>
<tr>
<td>mibench/sec/pgp/md5_encrypt</td>
<td>168 bytes</td>
</tr>
<tr>
<td>mibench/tele/fft</td>
<td>128 bytes</td>
</tr>
<tr>
<td>mibench/tele/fft_encrypt</td>
<td>16 bytes</td>
</tr>
<tr>
<td>mibench/tele/fft</td>
<td>24 bytes</td>
</tr>
<tr>
<td>mibench/tele/fft_encrypt</td>
<td>160 bytes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Manually Verified Stack Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>bsearch(x, l, h)</td>
<td>40(n + log₂(h_l - l_h)) bytes</td>
</tr>
<tr>
<td>fib(n)</td>
<td>24(n + 1) bytes</td>
</tr>
<tr>
<td>qsort(a, lo, hi)</td>
<td>48(h_l - lo + 1) bytes</td>
</tr>
<tr>
<td>filter_pos(a, sz, lo, hi)</td>
<td>48(h_l - l_h + 1) bytes</td>
</tr>
<tr>
<td>sum(a, lo, hi)</td>
<td>32(h_l + lo - 1) bytes</td>
</tr>
<tr>
<td>fact_log(n)</td>
<td>40 + 2n² bytes</td>
</tr>
</tbody>
</table>

Yale