Garbage Collection Makes Rust Easier to Use: A Randomized Controlled Trial of the Bronze Garbage Collector

Michael Coblenz, Michelle Mazurek, Michael Hicks
RUST: SAFER BUT HARD TO LEARN

- C, C++ allow dereferencing arbitrary pointers
- Chromium: > 70% of severe security bugs due to memory safety problems [Google]
- Rust is memory-safe (unlike C, C++)
- Ownership mechanism provides memory safety, avoids cost of GC
- Fulton et al.:
  - 59% of survey respondents: Rust is harder to learn than other languages
  - 7/16 interviewees: biggest challenges are ownership/borrowing
BRONZE: A NEW GC FOR RUST

- Idea: mitigate usability cost of ownership with a garbage collector
  - Most code is not performance-critical
  - Use garbage collector for most code
- GCs trace the heap to find live objects, starting from roots
- Modified Rust compiler to emit LLVM stack maps
  - Enables prototype's runtime to find roots automatically
DOES BRONZE HELP?

- Randomized controlled trial with 333 participants from a programming languages class
### PROCEDURE

- Four lectures on Rust
- Two live-coding demos (after students said the tasks were very hard)
- Survey after each task

<table>
<thead>
<tr>
<th>8 days</th>
<th>12 days</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topic</td>
<td>Traditional</td>
</tr>
<tr>
<td>Basics</td>
<td>Basics\textsubscript{noGC}</td>
</tr>
<tr>
<td>Ownership, lifetimes</td>
<td>Ownership\textsubscript{GC}</td>
</tr>
<tr>
<td>Aliased, mutable data</td>
<td>Aliasing\textsubscript{noGC}</td>
</tr>
<tr>
<td>Aliased, mutable data</td>
<td>(none)</td>
</tr>
</tbody>
</table>
No difference in completion rates of Ownership task

Bronze users were 2.4x as likely to score 100% on Aliasing \((p \approx .006)\)
RESULTS: COMPLETION TIMES

- Bronze: finished Aliasing faster (median 4 h. vs. 12 h., p < .001)
RESULTS: COMPLETION TIMES

- No significant difference in total times

<table>
<thead>
<tr>
<th>Topic</th>
<th>Traditional</th>
<th>Bronze</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basics</td>
<td>Basics&lt;sub&gt;noGC&lt;/sub&gt;</td>
<td>Basics&lt;sub&gt;noGC&lt;/sub&gt;</td>
</tr>
<tr>
<td>Ownership, lifetimes</td>
<td>Ownership&lt;sub&gt;noGC&lt;/sub&gt;</td>
<td>Ownership&lt;sub&gt;GC&lt;/sub&gt;</td>
</tr>
<tr>
<td>Aliased, mutable data</td>
<td>Aliasing&lt;sub&gt;noGC&lt;/sub&gt;</td>
<td>Aliasing&lt;sub&gt;GC&lt;/sub&gt;</td>
</tr>
<tr>
<td>Aliased, mutable data</td>
<td>(none)</td>
<td>Aliasing&lt;sub&gt;noGC&lt;/sub&gt;</td>
</tr>
</tbody>
</table>
Asked same question after each part

Participants were more likely to think GC was helpful after doing the assignment than before it ($p < .001$)
"HOW MUCH DO YOU LIKE RUST?"

- No significant difference

Diagram showing the distribution of responses for different categories with Ownership, Aliasing, and Ownership with and without GC, and a fraction scale from 0.00 to 1.00.
FACTORS INFLUENCING PERCEPTIONS

- Perhaps future designs should focus on reducing stress rather than time spent!

0 would mean "no correlation"
-1 or 1 would mean "can completely predict perception of time spent from amount liking Rust"
“WHAT WAS MOST DIFFICULT?”

“Coding with ownership rules and trying to implement mutability, in general, was just such a headache. It is like someone had combined the worst part of C and Java.”

Interior mutability requires, in API:

- Managing reference-counted pointers
- Using dynamic borrowing
- Whereas GC references "just work"
NEXT STEPS

▸ How can we make Rust even easier to use?

▸ Observed students in a Rust course

▸ Key opportunities:
  ▸ Low-level error messages do not teach high-level concepts
  ▸ Fixing compiler errors is like debugging. How to teach debugging effectively?
  ▸ Showing partial lists of errors is misleading
  ▸ Motivating understanding rather than cargo culting
DISCUSSION

- Reaction from the community
- What questions would YOU have asked?
- How would you have changed the study?
- Limitations?
- You could work on this project too!
CONCLUSION

- Garbage collection significantly reduces the architectural burden of ownership in Rust
- GC can enable completion of complex tasks in less time in ownership-based languages