RustProof

Drew Gohman, Matthew O’Brien, Bradley Rasmussen, Sami Sahli, Michael Salter, Vincent Schuster, Matthew Slocum

Sponsored by Aaron Tomb and Jamey Sharp
What is RustProof?

- Formally verifies user program correctness
- Used for “mission-critical” applications

Why RustProof?

- Testing and inspection aren’t exhaustive
- Zero runtime cost
Demo

http://viewpure.com/6AONwoGaquw?start=0&end=0
Constraints and Assumptions

Constraints
- Had to be a compiler plugin
- Had to use an unstable version of Rust (nightly)

Assumptions
- No platform-specific problems
- We could find a working library of Rust bindings to Z3
- Documentation about Rust nightlies was complete
- Reporting through compiler would be simple
Features

Accept user attributes on functions
Construct verification condition(s)
Pass verification condition(s) to a solver
Return output from solver, including counterexamples

Can’t function without all of these in place

Integer arithmetic
Assertions
Conditionals
Boolean arithmetic
...

Could be added incrementally
Deliverables

- Requirements specification document
- Risk management plan
- Software architecture design document
- Work breakdown list
- RustProof usage documentation
- RustProof
## Team Roles

<table>
<thead>
<tr>
<th>Member</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drew Gohman</td>
<td>IT, backups, GitHub</td>
</tr>
<tr>
<td>Matthew O’Brien</td>
<td>Workflow</td>
</tr>
<tr>
<td>Bradley Rasmussen</td>
<td>Backup team lead</td>
</tr>
<tr>
<td>Sami Sahli</td>
<td>Team lead</td>
</tr>
<tr>
<td>Michael Salter</td>
<td>Requirements</td>
</tr>
<tr>
<td>Vincent Schuster</td>
<td>Risk management</td>
</tr>
<tr>
<td>Matthew Slocum</td>
<td>Rust “master genius”</td>
</tr>
</tbody>
</table>
Process and Schedule

- Planning
- Requirements
- Risk Management
- Design
- Research
- Implementation
- Testing

External Applications

<table>
<thead>
<tr>
<th></th>
<th>GitHub</th>
<th>Travis</th>
<th>Slack</th>
<th>Backups</th>
</tr>
</thead>
</table>

# Problems and Contingencies

<table>
<thead>
<tr>
<th>Issue</th>
<th>Mitigation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brand new language</td>
<td>Individual research and specific team member focus</td>
</tr>
<tr>
<td>Linear workflow</td>
<td>Break into pieces where possible, and group coding on major components</td>
</tr>
<tr>
<td>Unit tests required overly-complicated structures</td>
<td>Prioritized extensive system tests to cover functionality</td>
</tr>
<tr>
<td>New unstable version broke system testing</td>
<td>Specified older version of nightly, began looking into other designs</td>
</tr>
<tr>
<td>Staying up to date with rapid changes in project</td>
<td>Weekly check-in with team members, and regular status updates</td>
</tr>
</tbody>
</table>
Lessons Learned

Working on top of an evolving language is hard
- Features are not sufficiently documented
- Language prone to change
- Completely unsupported dependencies

Finding resources in open source
- Googling works ok, until it doesn’t
- Reading documentation, then the source code
- Contacting Rust community and developers

Team dynamics, coding styles, and how to compromise
- Communicating problems without code shaming
- Establish code style guidelines early and stick to them
- Choosing battles over design choices
Questions?