Next steps for Checked C

DAVID TARDITI

LIGHTNING TALK, PACIFIC NORTHWEST WORKSHOP ON PROGRAMMING LANGUAGES AND SOFTWARE ENGINEERING

MAY 9, 2023

Checked C

- Extends C with type-safety
  - New pointer types: _Ptr, _Array_ptr, _Nt_array_ptr
  - Declare bounds for _Array_ptr
  - Null/bounds checking at runtime
  - Checked regions
    - Restrictions on unsafe pointers/casts.
  - Assumption: memory mgmt is correct.
- Includes language spec, compiler, and tools for translation
- Language spec/compiler developed at Microsoft from 2015-2021
- No real-world use, even though spatial safety remains a big problem!

Continues as open-source research project

- Fork at https://github.com/secure-sw-dev.
- Setup non-profit "Secure Software Development Project".
- Working with Aravind Machiry, Mike Hicks, John Criswell, and others.
- Recent contributions:
  - 3C tool for translating C to Checked C (Aravind Machiry, Mike Hicks and others). OOPSLA '22 (Distinguished Paper).
  - Fat-pointers for temporal safety (Jie Zhou, John Criswell, and Mike Hicks), Upcoming OOPSLA '23.
  - Support for erasing annotations for C compilers that don't support Checked C (in progress).
  - Experiments with converting open-source systems code.

Code must still compile with existing compilers.

- Checked C is backward-compatible: existing C code is valid (but unsafe).
- The Checked C extensions do not work with existing C compilers, though!
- Feedback/experience:
  - "But I own an open-source C library/RTOS/OS/utility. It needs to compile with other compilers. I can't add your annotations."
- Working on supporting erasure of annotations using macros.

Pursue more verification

- Introduced dynamic bounds checks and null-pointer checks.
  - A program terminates with a signals if check fails (could use signals to catch them).
- We think: This is great! No more undefined behavior! (PL centric view)
- Feedback:
  - "But now my program crashes and it might have still worked before. This is terrible!"
  - Programmers prefer program not crash at all due to these errors.
- They actually want **more** verification capabilities.

Add a static checking only mode

- We extended the dynamic semantics of the language.
- Now it only works with our modified clang compiler.
  - No one wants to switch compilers. Especially to a research-based compiler!
  - Linux uses GCC, Windows uses Visual C++.
- The necessary checks could be written by the programmer instead.
- Open question: how hard is it to write the checks/prove safety?

Languages and verification

- In the future, all languages for writing secure software will have verification built-in!
  - Verification technology is getting too good to ignore.
- Problem: existing widely-used languages don't have verification built-in.
- What to do about them?
  - One approach: retrofit verification technologies.
  - Some verification is better than none.
- Lessons learned from Checked C apply.

Lessons

- Make sure code compiles with existing compilers.
- Write stand-alone checker/verifier tools if possible.
  - Yes, you might miss some subtle aspects for a specific compiler. That can be fixed.
- Pursue more verification.
  - You'll be able to prove (some) properties programmers really care about.
  - Automatic formal verification that shades into deductive verification is fine.