Modular Development of Certified Concurrent Code

Pl: Zhong Shao  Student: Xinyu Feng  NSF CCF Grant 0524545
http://flint.cs.yale.edu

Goal: Building Reliable Software!

We have NO security guarantee without reliable implementation!
Low-level program verification allows us to have:
• Highly efficient and dependable OS and libraries
• Reliable security infrastructure
• Certified applications for mission-critical systems

First Step:
Modular Verification of Low-Level Concurrent Code!

Comparison to state of the art

Current approaches
• higher-level, cobegin/coend
• PCC for sequential code
• not modular or thread modular only
• type systems for specific properties

New approach
• low-level + fork/join/exit
• PCC for concurrent code
• thread modular and procedural modular
• logic system for general properties

Concurrent Verification

Assume-Guarantee Method
• Thread spec: (A, G)
• Non-interference: ∀i,j. ij ⇒ Gi ⇒ Aj

Challenges for low-level code cert.
• Changing Environment
  • Unbounded Dynamic Thread Creation/Termination
• Modularity Issues
  • Thread Modularity + Procedural Modularity

The CMAP Approach

Queue Update
WF(Q∪{t}, Θ∪{(A, G)}))

Queue Extension
WF(Q∪{t}, Θ∪{(A, G)}))

INV: dynamic threads do not interfere
Queue: Q, Spec: Θ,
WF(Q, Θ) = ∀i,j. i≠j ⇒ Xi ⇒ Ai

Borrow the invariant-based proof in TAL
Initial condition: ∃Θ0. WF(Q0, Θ0)

Conclusions and Future Work

What we have done so far:
CMAP: Abstract Machine + Verification Logic
Certified Code:
Unbounded dynamic thread creation
Readers-writers problems with sync. primitives
Lock-free concurrent programs (GCD)
Coq Implementation

Future Work
Certified thread library and sync. primitives
Surface languages and certifying compilation

Foundational Proof-Carrying Code (FPCC)

Certifying Toolkit
Spec Assembly Code
Proof Checker

Safety Proof

Certify once, use everywhere!