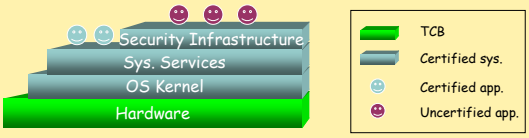
Modular Development of Certified Concurrent Code

PI: 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!

Our Approach:

Modular Verification of Low-Level Multi-Threaded Code (CMAP)

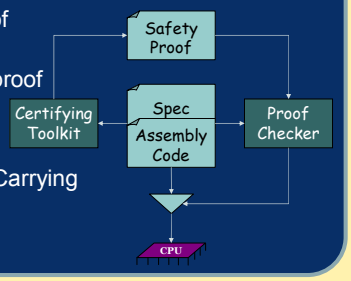
Assume-Guarantee + Dynamic Thread Creation

Logic-based specification

Semi-automatic proof construction

Formal Mechanical proof

Foundational Proof-Carrying Code (FPCC)



Comparison to state of the art

Current approaches	New approach
• higher-level, cobegin/coend	• low-level + fork/join/exit
• PCC for sequential code	• PCC for concurrent code
• not modular or thread modular only	• thread modular and procedural modular
• type systems for specific properties	• logic system for general properties

Concurrency Verification

Assume-Guarantee Method

- Thread spec: (A, G)
- Non-interference: $\forall i,j.\ i\neq j \Rightarrow G_i \Rightarrow A_j$

Challenges for low-level code cert.

- Changing Environment
  - Unbounded Dynamic Thread Creation/Termination
- Modularity Issues
  - Thread Modularity + Procedural Modularity

```
Global data: int data[100];
main:
  int i:=0;
  while (i<100){
    data[i]:=f(i);
    fork child(i);
    i++;
  }
```

The CMAP Approach

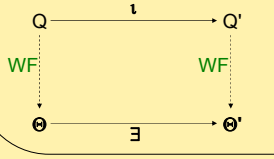
INV: dynamic threads do not interfere

Queue: **Q**, Spec: **Θ**,

$WF(Q, \Theta) \equiv \forall t_i, t_j \in Q. t_i \neq t_j \Rightarrow (G_i \Rightarrow A_j)$

Borrow the invariant-based proof in TAL

Initial condition: $\exists \Theta_0.\ WF(Q_0, \Theta_0)$



ι ::= add | sub | jd …

| exit | fork | yield

The CMAP Approach

Queue Update

WF(Q∪{t}, Θ∪{(A, G)})

↓ A⇒A', G'⇒G; t: (A', G')

WF(Q∪{t}, Θ∪{(A', G')})

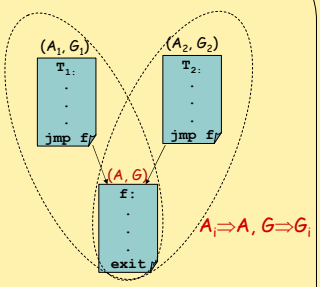
Queue Extension

WF(Q∪{t}, Θ∪{(A, G)})

↓ fork f(a)

↓ A ⇒ A", G" ⇒ G

WF(Q∪{t',t}, Θ∪{(A", G"), (A∨G", G∧A")})



Certify once, use everywhere!

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

National Science Foundation WHERE DISCOVERIES BEGIN

NSF Cyber Trust Annual Principal Investigator Meeting
Sept. 25 -27th 2005
Newport Beach California

The FLINT Project
Systems Software for Advanced Languages