Our Research on Verified Software --Compilation and Verification

Yi-yun Chen, Yu Zhang, Yu Guo, Zhao-peng Li, ......

USTC-Yale Joint Research Center for High-Confidence Software
School of Computer Science and Technology
University of Science and Technology of China

October, 2009
Outline

- Overview of Our Center
- Several Research Projects of Our Center
  - Certified Operating System Kernels
  - Concurrent Software Verification
  - Certifying Compiler
Overview of Our Center

USTC-Yale Joint Research Center for High-Confidence Software

http://kyhcs.ustcsz.edu.cn

- Founded in Oct. 2008
- Cooperation with Prof. Zhong Shao (Flint, Yale): more than 5 years

- **Research on High-Confidence Software**
  - Develop new cutting-edge technologies and tools improving the **dependability** of software
Overview of Our Center

Our Researches

- Annotated Source Code
- Certifying Compiler
- Low-level Code
- Proof Checker
- Proof
- Verification Framework

OS Kernels

Concurrent Software

By Manual
Overview of Our Center

Our Approach to Verification

Calculus of Inductive Constructions (CiC), Coq
Overview of Our Center

Our Approach to Verification

Typical x86/MIPS Machine model
Overview of Our Center

Our Approach to Verification

To link modules certified by different domain specific logics
Overview of Our Center

Our Approach to Verification

OS Kernels:
Boot-loader,......

Concurrent Software:
reentrant locks,......

Certifying Compiler:
pointer logic,....
Outline

- Overview of Our Center

- Several Research Projects of Our Center
  - Certified Operating System Kernels
  - Concurrent Software Verification
  - Certifying Compiler
Outline

- Overview of Our Center

- Several Research Projects of Our Center
  - Certified Operating System Kernels
  - Concurrent Software Verification
  - Certifying Compiler
Motivation

**Why we choose OS kernels**

- **OS kernels need be verified**
  - Low-level computing platform
  - Complicated
  - Lurking bugs

- **Academic interests**
  - Rich features
  - Verification at different abstraction layers

http://kyhcs.ustcsz.edu.cn/certos
Previous work

- **A fully-certified mini-kernel prototype**
  - Joint work with
    - Xinyu Feng (Yale), Zhong Shao (Yale), Yuan Dong (Tsinghua Univ.)
  - 150 lines of assembly code (runnable)
    - Thread scheduling with timer interrupt
    - Synchronization primitives
  - > 82,000 lines of Coq proof scripts
    - (by hand)
    - Definitions, lemmas, theorems & proof

**Limitation**
- Concurrent user code + non-preemptive kernel
On-going Work

Concurrent user code
- lock
- spawn
- yield
- exit

Preemptive kernel
- scheduler
- new
- scheduler

Context switching routine
- Context switching

General Logic
- Concurrent Separation Logic
- New Logic
- Sequential Hoare Logic
Outline

- Overview of Our Center
- Several Research Projects of Our Center
  - Certified Operating System Kernels
  - Concurrent Software Verification
  - Certifying Compiler
Concurrent Software Verification

**Motivation**

The adoption of multi-core computer platform accelerates the requirements for developing (shared-resource-based) concurrent software!

**Research Goals**

- **Long-term goal**: verification methods on verifying concurrent software
- **Short-term goal**: focusing on verifying low-level code using various synchronization mechanisms

VSysc – Verifying Synchronization mechanisms

http://kyhcs.ustcsz.edu.cn/vsync
Concurrent Software Verification

Previous Work

Reason about low-level code using ...

- Locks
  - Reentrant locks/ Read-write locks
  - Extend Concurrent Separation Logic (CSL)

- Transactional Memory (TM)
  - TM1: an abstract model with big commit primitive
    - Invariant-based proof method
  - TM2: a lazy STM model with thin primitives (splitting big commit primitive)
    - CSL + permission accounting in separation logic
Concurrent Software Verification

**Ongoing Work**

- To research the semantics for safely integrating various software TM implementations and locking

General Intermediate Language

- Compilation Techniques: source->IR
- Program logics: IR, *impl.*
- Implementation Techniques: IR->target code

To express both locking and STMs
Outline

❖ Overview of Our Center

❖ Several Research Projects of Our Center
  ➢ Certified Operating System Kernels
  ➢ Concurrent Software Verification
  ➢ Certifying Compiler
Research on Certifying Compiler

**Motivation**

- **Connect source and target verification**
  - Source-level verification is more friendly
  - Low-level verification is fundamental

- **Make verification more productive**
  - Getting benefits from the certifying compiler
Previous Work

A certifying compiler for PointerC

PointerC is a subset of C programming language

- A pointer logic: an extension of Hoare logic
  - to reason about programs with (circular) singly-linked list, (circular) doubly-linked list or tree

- Methods for producing low-level proof

- A low-level framework
  - including machine and logic system
Previous Work (cont.)

Certifying Compiler for PointerC

Annotated PointerC Code

Verification Condition Generator (source-level)

VC Proof Translator & Target Proof Generator

Certifying Compiler for PointerC

Annotated x86 Assembly Code

Proof assistant Coq

Verification Condition (VC)

VC Proof
On-going Work: Project CComp

- **Current Objectives**
  - Explore techniques on integrating several (automated) theorem provers in the certifying compiler
  - Develop techniques of automated theorem proving with proof-term output
  - Research proof generation techniques for a more realistic target program logic
Project CComp: Overview

C-like language + Separation Logic

/*n>0 | emp*/
struct list*
list_create(int n) {
    ...
    while(n>0)
    /*n>=0 | list(p)*/
    {... }
    return p;
}
/* true | list(res)*/

CComp Certifying Compiler

Low-level Verification Framework

List_create:
push ebp
mov esp, ebp
sub esp, 8
jmp L0
L0 :
...
L1 :

Lemma wf_L0:
...
Lemma wf_L1:
...
Lemma ...

USTC-Yale Joint Research Center for HCS
Project CComp : Framework

- **Annotated PointerC Code**
- **Verification Condition Generator (source-level)**
  - **VC Proof Translator & Target Proof Generator**
  
- **VC Proof Translator & Target Proof Generator**

**CComp Certifying Compiler**

- **Automated Theorem Prover**
  
- **Annotated x86 Assembly Code**

**Proof**
Automated Theorem Prover

Formulas

Prover for Separation Logic Fragments

Prover for Linear Integer Arithmetic

Other Domain-Specific Provers (list, and etc.)

Proof Tree

Proof Output

Proof (Proof Term Checkable by Coq)

Separation Logic Fragment: separation conjunction/emp/p|->_
Built-in Predicates: list/lseg/dlist/dlseg/tree...
Automated Theorem Prover

1. Based on Simplex decision procedure;
2. Provides interfaces for VC Gengerator and other provers.

Formulas

Prover for Separation Logic Fragments

Prover for Linear Integer Arithmetic

Other Domain-Specific Provers (list, and etc.)

Proof Tree

Proof Output

Proof (Proof Term Checkable by Coq)

Separation Logic Fragment : separation conjunction/ emp / p|->_
Built-in Predicates : list/lseg/dlist/dlseg/tree...
Automated Theorem Prover

1. Inspired by Smallfoot;
2. Supports logic fragments including separation conjunction and built-in predicates (list/lseg/dlist/dlseg/tree...).

Separation Logic Fragment: separation conjunction/ emp / p|->_
Built-in Predicates: list/lseg/dlist/dlseg/tree...
Proof Translation & Generation

- **Features on Specification Translation**
  - Pre-/post-conditions of functions and loop invariants are translated from source level.
  - Other assertions are generated according to low-level inference rules.

- **Features on Proof Generation**
  - Proof terms are constructed to show the well-formedness of each basic block.
  - The process uses Coq proof templates and tactics.
  - Some proofs are produced by the built-in automated theorem provers.

```coq
{p} L: add ebx, eax {p'}
... jmp...

Proof.
... Qed.
```
Currently 3 person-years

- **30k LoC SML/NJ**
  - Parser, Type Checker, VC Generator, Prover, Proof Generator ...

- **16k LoC Coq**
  - Prover libraries, Low-level verification framework, Auxiliary tactics