Towards Certified Separate Compilation for Concurrent Programs

Hanru Jiang*   Hongjin Liang†
Siyang Xiao*   Junpeng Zha*   Xinyu Feng†

* University of Science and Technology of China
† Nanjing University
Compilers are *NOT* Trustworthy

Finding and Understanding Bugs in C Compilers

Xuejun Yang  Yang Chen  Eric Eide  John Regehr
University of Utah, School of Computing
{jxyang, chenyang, eeide, regehr}@cs.utah.edu

[PLDI 2011]

- 11 open-source/commercial compilers were tested
- Found **325** bugs, in **EVERY** compiler!
Compilers are **NOT** Trustworthy

*Finding and Understanding Bugs in C Compilers*

Xuejun Yang, Yang Chen, Eric Eide, John Regehr

University of Utah, School of Computing

{jxyang, chenyang, eeide, regehr}@cs.utah.edu

[PLDI 2011]

- 11 open-source/commercial compilers were tested
- Found **325** bugs, in **EVERY** compiler!

**Verification of compiler correctness helps:**

“The striking thing about our **CompCert** results is that the **middle end bugs** we found in all other compilers are absent.”
Compilation Correctness

Source (e.g. C) \[ S \]

Compiler

Target (e.g. assembly) \[ T \]

\[ \forall S, T . \ T = \text{Compiler}(S) \implies T \subseteq S \]

Correct(Compiler) :

Semantic preservation:

T has no more observable behaviors (e.g. I/O events by print) than S.
Compiler Verification
Compiler Verification

• Leroy’06: Formal certification of a compiler back-end
• Lochbihler’10: Verifying a compiler for Java threads
• Myreen’10: Verified just-in-time compiler on x86
• Sevcik et al.’11: Relaxed-memory concurrency and verified compilation
• Zhao et al.’13: Formal verification of SSA-based optimizations for LLVM
• Kumar et al.’14: CakeML: A verified implementation of ML
• Stewart et al.’15: Compositional CompCert
• Kang et al.’16: Lightweight Verification of Separate Compilation
• …
Compiler Verification

- Leroy’06: Formal certification of a compiler back-end
- Lochbihler’10: Verifying a compiler for Java threads
- Myreen’10: Verified just-in-time compiler on x86
- Sevcik et al.’11: Relaxed-memory concurrency and verified compilation
- Zhao et al.’13: Formal verification of SSA-based optimizations for LLVM
- Kumar et al.’14: CakeML: A verified implementation of ML
- Stewart et al.’15: Compositional CompCert
- Kang et al.’16: Lightweight Verification of Separate Compilation
- ...

*Limited support of separate compilation and concurrency!*
Real-world programs may consist of multiple components, which will be compiled independently.
Real-world programs may consist of multiple components, which will be compiled independently.
Separate Compilation

Source (e.g. C)

S1

Interaction

S2
Separate Compilation

Source (e.g. C) → Interaction → Target (e.g. assembly)

Compiler-1

S1 → Interaction → S2

T1
Separate Compilation

Source (e.g. C) → S1 → Interaction → S2 → Target (e.g. assembly)

Compiler-1 → T1

Compiler-2 → T2
Separate Compilation

Source (e.g. C)

Compiler-1

Target (e.g. assembly)

Compiler-2
Separate Compilation

Source (e.g. C)

Compiler-1

Interaction

Different compilers

Compiler-2

Target (e.g. assembly)
Separate Compilation

Source (e.g. C)

Different languages

S1 <-> S2

Compiler-1

Different compilers

T1 <-> T2

Compiler-2

Target (e.g. assembly)
Separate Compilation of Concurrent Programs

Source (e.g. C) → Compiler-1 → Target (e.g. assembly) → Compiler-2 → S1

S1 and S2 communicate via Interaction.

T1 and T2 communicate via Interaction.
Separate Compilation of **Concurrent** Programs

Source (e.g. C)

**S1**

Compiler-1

Target (e.g. assembly)

T1

Parallel Composition ||

**S2**

Compiler-2

T2
Can we reuse existing certified compilers (e.g. CompCert) for separate compilation of concurrent programs?
Compositional CompCert’s Argument...

[Stewart et al. POPL’15]

YES, for data-race-free (DRF) concurrent programs
Intuition of the Argument:
Interleaving <=> Non-preemptive for DRF Programs

```plaintext
r1 = 1;
r1 = r1 + 1;
lock();
x = 1;
y = x + 1;
unlock();

r2 = 2;
r2 = r2 + 1;
lock();
x = 2;
y = x + 1;
unlock();
```

interleaving
Intuition of the Argument:
Interleaving $\iff$ Non-preemptive for DRF Programs

```
r1 = 1;
r1 = r1 + 1;
lock();
x = 1;
y = x + 1;
unlock();

r2 = 2;
r2 = r2 + 1;
lock();
x = 2;
y = x + 1;
unlock();
```

No race

interleaving
Intuition of the Argument:
Interleaving $\iff$ Non-preemptive for DRF Programs

```
// Thread 1
r1 = 1;
r1 = r1 + 1;
lock();
x = 1;
y = x + 1;
unlock();

// Thread 2
r2 = 2;
r2 = r2 + 1;
lock();
x = 2;
y = x + 1;
unlock();
```

Interleaving

No race
Intuition of the Argument:
Interleaving $\iff$ Non-preemptive for DRF Programs

$$r_1 = 1;$$
$$r_1 = r_1 + 1;$$
$$\text{lock();}$$
$$x = 1;$$
$$y = x + 1;$$
$$\text{unlock();}$$

$$r_2 = 2;$$
$$r_2 = r_2 + 1;$$
$$\text{lock();}$$
$$x = 2;$$
$$y = x + 1;$$
$$\text{unlock();}$$

No race
sequential
$$r_1 = 1;$$
$$r_1 = r_1 + 1;$$
$$\text{yield;}$$
$$x = 1;$$
$$y = x + 1;$$
$$\text{yield;}$$
sequential
$$r_2 = 2;$$
$$r_2 = r_2 + 1;$$
$$\text{yield;}$$
$$x = 2;$$
$$y = x + 1;$$
$$\text{yield;}$$

Non-preemptive: yield control at certain points only
Intuition of the Argument:
Interleaving $\iff$ Non-preemptive for DRF Programs

Plausible, but need to address several key challenges
Challenges
Challenges

• How to formulate DRF in language independent manner?
Challenges

• How to formulate **DRF** in **language independent** manner?

• How to prove **DRF-preservation**, compositionally?

![Diagram showing DRF concept with S1, S2, T1, and T2]

**DRF**
Challenges

• How to formulate **DRF** in **language independent** manner?

• How to prove **DRF-preservation**, compositionally?
Challenges

• How to formulate **DRF** in **language independent** manner?

• How to prove **DRF-preservation**, compositionally?
Challenges

• How to formulate DRF in language independent manner?

• How to prove DRF-preservation, compositionally?
Challenges

• How to formulate DRF in language independent manner?

• How to prove DRF-preservation, compositionally?

• How to support benign-race and relaxed memory models?
Challenges

• How to formulate DRF in language independent manner?

• How to prove DRF-preservation, compositionally?

• How to support benign-race and relaxed memory models?

“… synchronization primitives are commonly implemented with assembly code that has data races.”

—— Hans-J. Boehm, HotPar’11
Our Work
Our Work

• **Language independent** verification framework
  - Key semantics components + proof structures
  - Supports separate compilation for **race-free** concurrent programs
    - With both external function calls & multi-threaded code
Our Work

• **Language independent** verification framework
  - Key semantics components + proof structures
  - Supports separate compilation for **race-free** concurrent programs
    - With both external function calls & multi-threaded code
• Framework extension:
  - Supports **x86-TSO + confined benign-races**
Our Work

- **Language independent** verification framework
  - Key semantics components + proof structures
  - Supports separate compilation for **race-free** concurrent programs
    - With both external function calls & multi-threaded code
- Framework extension:
  - Supports **x86-TSO** + **confined benign-races**
- **CASCompCert**:
  - Extends CompCert with **Concurrency** + **Abstraction** + **Separate compilation**
  - **Reuses** considerable amount of CompCert proofs
  - **Racy** x86-TSO impl. of locks as synchronization library
Outline of this Talk

• Language-independent DRF formulation
• DRF-preservation and key proof structures
• Supporting x86-TSO and confined benign-races in CASCompCert
Outline of this Talk

• Language-independent DRF formulation

• DRF-preservation and key proof structures

• Supporting x86-TSO and confined benign-races in CASCompCert
Language-Independent DRF

Data-race: read-write / write-write conflicts
Language-Independent DRF

Data-race: read-write / write-write conflicts
Language-Independent DRF

Data-race: read-write / write-write conflicts

*Why language-independent?*
  To support cross-language interaction

May in different languages
Language-Independent DRF

Data-race: read-write / write-write conflicts

Why language-independent?
To support cross-language interaction
abstract away lang. details
e.g. interaction semantics

May in different languages

[Stewart et al. POPL’15]
Language-Independent DRF

Data-race: read-write / write-write conflicts

Why language-independent?
To support cross-language interaction
abstract away lang. details
e.g. interaction semantics

May in different languages

[Stewart et al. POPL’15]

NO concrete reads/writes

Semantics \((G \ C \ M : \text{Type}) : \text{Type} \triangleq \)
- \(\text{initial\_core} : G \to \mathcal{V} \to \text{list } \mathcal{V} \to \text{option } C\)
- \(\text{at\_external} : C \to \text{option } (\mathcal{F} \times \text{list } \mathcal{V})\)
- \(\text{after\_external} : \text{option } \mathcal{V} \to C \to \text{option } C\)
- \(\text{halted} : C \to \text{option } \mathcal{V}\)
- \(\text{corestep} : G \to C \to M \to C \to M \to \text{Prop}\)
Language-Independent DRF

Data-race: read-write / write-write conflicts

Why language-independent?

How to formulate DRF

if we do not even know the concrete reads/writes?

May in different languages

NO concrete reads/writes

Semantics $(G, C, M : \text{Type}) : \text{Type} 
\begin{align*}
\text{initial_core} & : G \rightarrow \mathcal{V} \rightarrow \text{list} \mathcal{V} \rightarrow \text{option} C \\
\text{at_external} & : C \rightarrow \text{option} (\mathcal{F} \times \text{list} \mathcal{V}) \\
\text{after_external} & : \text{option} \mathcal{V} \rightarrow C \rightarrow \text{option} C \\
\text{halted} & : C \rightarrow \text{option} \mathcal{V} \\
\text{corestep} & : G \rightarrow C \rightarrow M \rightarrow C \rightarrow M \rightarrow \text{Prop}
\end{align*}
Solution: Abstract Footprints

\[ \text{DRF}(S1 \| S2) \]
Solution: Abstract Footprints

Defined in terms of footprint disjointness

$\text{DRF}(S_1 \parallel S_2)$

- footprints $\delta ::= (rs, ws)$
  - read-set
  - write-set
Solution: Abstract Footprints

- Defined in terms of footprint disjointness
  
  \[ \text{DRF}(S_1 \parallel S_2) \]

- footprints \( \delta ::= (rs, ws) \)

- well-defined language
  
  extensional characterization of footprints
Solution: Abstract Footprints

Defined in terms of footprint disjointness

DRF(S1 || S2)

• footprints $\delta ::= (rs, ws)$

• well-defined language
  extensional characterization of footprints
Solution: Abstract Footprints

Defined in terms of footprint disjointness:

$$\text{DRF}(S1 \parallel S2)$$

- footprints $\delta ::= (rs, ws)$
- well-defined language:
  extensional characterization of footprints
Solution: Abstract Footprints

- Defined in terms of footprint disjointness

\[ \text{DRF}(S1 || S2) \]

- footprints \( \delta ::= (\text{rs}, \text{ws}) \)
  - read-set
  - write-set

- well-defined language
  - extensional characterization of footprints

Arbitrarily different
Solution: Abstract Footprints

- Defined in terms of footprint disjointness
  \[ \text{DRF}(S_1 \parallel S_2) \]

- footprints \( \delta ::= (\text{rs}, \text{ws}) \)
  - read-set
  - write-set

- well-defined language
  extensional characterization of footprints

Arbitrarily different
Solution: Abstract Footprints

Defined in terms of footprint disjointness

\[ \text{DRF}(S_1 \parallel S_2) \]

- Footprints \( \delta := (rs, ws) \)

- Well-defined language
  
  Extensional characterization of footprints

Arbitrarily different

Same
Solution: Abstract Footprints

- Defined in terms of footprint disjointness
  \[ \text{DRF}(S1 \parallel S2) \]

- footprints \( \delta ::= (rs, ws) \)
  - read-set
  - write-set

- well-defined language
  extensional characterization of footprints
Solution: Abstract Footprints

**Definition 1** (Well-Defined Languages). \( \text{wd}(tl) \) iff, for any execution step \( F \vdash (\kappa, \sigma) \xrightarrow{\delta} (\kappa', \sigma') \) in this language, all of the following hold (some auxiliary definitions are in Fig. 6):

1. \( \text{forward}(\sigma, \sigma') \);
2. \( \text{LEffect}(\sigma, \sigma', \delta, F) \);
3. For any \( \sigma_1 \), if \( \text{LEqPre}(\sigma, \sigma_1, \delta, F) \), then there exists \( \sigma'_1 \) such that \( F \vdash (\kappa, \sigma_1) \xrightarrow{T}{\delta} (\kappa', \sigma'_1) \) and \( \text{LEqPost}(\sigma', \sigma'_1, \delta, F) \).
4. Let \( \delta_0 = \bigcup\{ \delta \mid \exists \kappa', \sigma'. F \vdash (\kappa, \sigma) \xrightarrow{\delta} (\kappa', \sigma') \} \). For any \( \sigma_1 \), if \( \text{LEqPre}(\sigma, \sigma_1, \delta_0, F) \), then for any \( \kappa'_1, \sigma'_1, \iota_1, \delta_1, F \vdash (\kappa, \sigma_1) \xrightarrow{T}{\delta_1} (\kappa'_1, \sigma'_1) \implies \exists \sigma'. F \vdash (\kappa, \sigma) \xrightarrow{T}{\delta_1} (\kappa'_1, \sigma'). \)

**Defined in terms of footprint disjointness**

**DRF(S1 \parallel S2)**

- footprints \( \delta ::= (rs, ws) \)
  - read-set
  - write-set

- well-defined language
  - extensional characterization of footprints
Outline of this Talk

- Language-independent DRF formulation
- DRF-preservation and key proof structures
- Supporting x86-TSO and confined benign-races in CASCompCert
Outline of this Talk

• Language-independent DRF formulation

• DRF-preservation and key proof structures

• Supporting x86-TSO and confined benign-races in CASCompCert
Compositional CompCert’s Argument…

\[ T_1 \parallel T_2 \subseteq S_1 \parallel S_2 \]

\[ \text{DRF}(S_1 \parallel S_2) \quad T_1 = \text{Comp}(S_1) \quad T_2 = \text{Comp}(S_2) \]
Compositional CompCert’s Argument…

\[ T_1 \parallel T_2 \subseteq S_1 \parallel S_2 \]

\[ \text{DRF}(S_1 \parallel S_2) \]

\[ T_1 = \text{Comp}(S_1) \quad T_2 = \text{Comp}(S_2) \]

Non-preemptive
Compositional CompCert’s Argument...

\[ T_1 \parallel T_2 \subseteq S_1 \parallel S_2 \]

\[ T_1 \| T_2 \subseteq S_1 \| S_2 \]

\[ \text{DRF}(S_1 \| S_2) \]

\[ T_1 = \text{Comp}(S_1) \quad T_2 = \text{Comp}(S_2) \]

Non-preemptive
Our Ideas

\[ T_1 \parallel T_2 \subseteq S_1 \parallel S_2 \]

DRF(S1 || S2)  

T1 = Comp(S1)  

T2 = Comp(S2)

Non-preemptive
Our Ideas

$T_1 \parallel T_2 \subseteq S_1 \parallel S_2$

$T_1 \setminus T_2 \subseteq S_1 \setminus S_2$

$DRF(S_1 \parallel S_2)$

$T_1 = \text{Comp}(S_1)$

$T_2 = \text{Comp}(S_2)$

Non-preemptive
Our Ideas

\[
T_1 \parallel T_2 \subseteq S_1 \parallel S_2 \subseteq \text{Trivial}
\]

\[
T_1 \mid T_2 \subseteq S_1 \mid S_2 \subseteq \text{Non-preemptive}
\]

\[
\text{DRF}(S_1 \parallel S_2)
\]

\[
T_1 = \text{Comp}(S_1) \quad T_2 = \text{Comp}(S_2)
\]
Our Ideas

\[ T_1 \parallel T_2 \subseteq S_1 \parallel S_2 \]

\[ T_1 \mid T_2 \subseteq S_1 \mid S_2 \]

\[ \text{DRF}(S_1 \parallel S_2) \]

\[ T_1 = \text{Comp}(S_1) \]

\[ T_2 = \text{Comp}(S_2) \]

Non-preemptive
Our Ideas

\[ \text{DRF}(T_1 \parallel T_2) \]

\[ T_1 \parallel T_2 \subseteq S_1 \parallel S_2 \]

\[ T_1 \mid T_2 \subseteq S_1 \mid S_2 \]

\[ \text{DRF}(S_1 \parallel S_2) \]

\[ T_1 = \text{Comp}(S_1) \]

\[ T_2 = \text{Comp}(S_2) \]

Non-preemptive
Our Ideas

DRF(T1 || T2) → T1 || T2 ⊆ S1 || S2

T1 | T2 ⊆ S1 | S2

DRF(S1 || S2)

T1 = Comp(S1)          T2 = Comp(S2)

Non-preemptive

Trivial
Our Ideas

T1 \succeq S1 \succeq \text{Trivial} \\
T1 \parallel T2 \succeq S1 \parallel S2

\text{DRF}(T1 \parallel T2) \text{DRF}(S1 \parallel S2)

T1 = \text{Comp}(S1) \quad T2 = \text{Comp}(S2)

\text{Non-preemptive}
Our Ideas

T1 || T2 ⊆ S1 || S2
T1 | T2 ⊆ S1 | S2

How to prove DRF-preservation?

T1 = Comp(S1)  T2 = Comp(S2)
Our Ideas

\[ T_1 \parallel T_2 \subseteq S_1 \parallel S_2 \]

\[ T_1 = \text{Comp}(S_1) \quad T_2 = \text{Comp}(S_2) \]

\[ \text{DRF}(T_1 \parallel T_2) \]

\[ \text{DRF}(S_1 \parallel S_2) \]
Our Ideas

T1 | T2 ⊆ S1 | S2 ⊆ Trivial

DRF(T1 || T2)

T1 || T2 ⊆ S1 || S2

T1 | T2 ⊆ S1 | S2

T1 | T2 ≲ S1 | S2

Footprint-preserving simulation

T1 = Comp(S1)

T2 = Comp(S2)
Our Ideas

\[ \text{DRF}(T_1 \parallel T_2) \]

\[ \text{DRF}(S_1 \parallel S_2) \]

T1 = Comp(S1) \quad T2 = Comp(S2)

Footprint-preserving simulation
Our Ideas

\[ T_1 \parallel T_2 \subseteq S_1 \parallel S_2 \]

\[ T_1 \parallel T_2 \subseteq S_1 \parallel S_2 \]

\[ T_1 = \text{Comp}(S_1) \quad T_2 = \text{Comp}(S_2) \]

Footprint-preserving simulation

DRF preservation

DRF(T₁ || T₂) ≤ S₁ || S₂
Our Ideas

Footprint-preserving simulation

Compositionality

DRF(T1 || T2) ⊆ S1 || S2

T1 | T2 ⊆ S1 | S2

T1 | T2 ≲ S1 | S2

T1 ≲ S1 ∧ T2 ≲ S2

T1 = Comp(S1)       T2 = Comp(S2)

DRF(S1 || S2)

DRF preservation
Our Ideas

\[ T_1 \cup T_2 \subseteq S_1 \cup S_2 \]

\[ T_1 \cup T_2 \subseteq S_1 \cup S_2 \trianglelefteq T_1 \cup T_2 \subseteq S_1 \cup S_2 \]

\[ T_1 = \text{Comp}(S_1) \]

\[ T_2 = \text{Comp}(S_2) \]

DRF preservation

Footprint-preserving simulation

Compositionality

DRF preservation

Our compiler correctness
Solution: Footprint-Preserving Simulation

Target has smaller footprints, so cannot introduce more races
Solution: Footprint-Preserving Simulation

\[
(S, \Sigma) \sim (T, \sigma)
\]

Target has smaller footprints, so cannot introduce more races
Solution: Footprint-Preserving Simulation

\[(S, \Sigma) \leq (T', \sigma')\]

Target has smaller footprints, so cannot introduce more races
Solution: Footprint-Preserving Simulation

Zero-or-multiple steps

Target has smaller footprints, so cannot introduce more races
Solution: Footprint-Preserving Simulation

Zero-or-multiple steps

Target has smaller footprints, so cannot introduce more races
Solution: Footprint-Preserving Simulation

Target has smaller footprints, so cannot introduce more races
Solution: Footprint-Preserving Simulation

$\Delta \downarrow \delta \downarrow \Delta, \delta: \text{Footprints}$

Target has smaller footprints, so cannot introduce more races
Solution: Footprint-Preserving Simulation

Target has smaller footprints, so cannot introduce more races
Outline of this Talk

- Language-independent DRF formulation
- DRF-preservation and key proof structures
- Supporting x86-TSO and confined benign-races in CASCompCert
Outline of this Talk

- Language-independent DRF formulation
- DRF-preservation and key proof structures
- Supporting x86-TSO and confined benign-races in CASCompCert
DRF Imposes Strong Restriction on Libraries
DRF Imposes Strong Restriction on Libraries

Client

Lib

T1 || T2

Call

Call

DRF

lock_rel:
...

mov $1, %eax

lock xchg %eax, L
...

Mem

mov $1, %eax

lock xchg %eax, L
DRF Imposes Strong Restriction on Libraries

Client

T1  \( \parallel \)  T2

Lib

DRF

lock_rel:
...
mov $1, %eax
lock xchg %eax, L
...

[spin-lock impl. in Linux 2.6]

\( \text{inefficient} \)

\( \text{inefficient} \)
DRF Imposes Strong Restriction on Libraries

[spin-lock impl. in Linux 2.6]
DRF Imposes Strong Restriction on Libraries

Client

T1

||

T2

Lib

DRF

lock_rel:

...  

mov $1, %eax

lock xchg %eax, L

...

Racy

lock_rel:

...  

mov $1, L

...

[spin-lock impl. in Linux 2.6]

Relaxed memory model, e.g. x86-TSO

inefficient

Inefficient, Racy

[spin-lock impl. in Linux 2.6]
DRF Imposes Strong Restriction on Libraries

Client

T1 || T2

Lib

DRF

Racy

lock_rel:

...  

mov $1, %eax

lock xchg %eax, L

...

[spin-lock impl. in Linux 2.6]

inefficient

Relaxed memory model, e.g. x86-TSO

mov $1, L

...

?
Our Idea
Our Idea

- Confined benign-races:
  - Racy libraries and client code run in separate memory regions
Our Idea

- Confined benign-races:
  - Racy libraries and client code run in separate memory regions
  - Client code be well-synchronized
Our Idea

• Confined benign-races:
  • Racy libraries and client code run in separate memory regions
  • Client code be well-synchronized
  • Racy libraries have race-free abstraction

Client

Well-synchronized

Call

T1

T2

Racy

Lib′_{Racy}

Race-free abstraction

Lib_{RF}

Call

T1

T2

Call

DRF
Supporting Confined Benign-Race & x86-TSO in Our CASCompCert
Supporting Confined Benign-Race & x86-TSO in Our CASCompCert

Source P

Clight

...  

Multi-threaded

Call

Lock
Supporting Confined Benign-Race & x86-TSO in Our CASCompCert

Source P

Clight

Multi-threaded

Call

Lock

race-free abstraction of spin-locks for synchronization
Supporting Confined Benign-Race & x86-TSO in Our CASCompCert

Source: Clight

Multi-threaded

Call

Lock

race-free abstraction of spin-locks for synchronization

DRF
Supporting Confined Benign-Race & x86-TSO in Our CASCompCert
Supporting Confined Benign-Race & x86-TSO in Our CASCompCert

Source

Clight
CompCert

Call
Lock

Target

x86

Race-free

DRF
Supporting Confined Benign-Race & x86-TSO in Our CASCompCert

Source P

CompCert

Clight

Call

DRF

Race-free

Target P

x86

Call

Lock

identity
Supporting Confined Benign-Race & x86-TSO in Our CASCompCert

Source $P$

CompCert

Target $P$

x86

Clight

Call

Lock

Race-free

Call

Lock

identity

x86-SC

DRF
Supporting Confined Benign-Race & x86-TSO in Our CASCompCert

Source P

- Clight
- CompCert
- x86

Target P

- Call
- DRF
- Race-free
- identity
- x86-SC
- Manually impl.
- Lock-Impl

with benign-races
Supporting Confined Benign-Race & x86-TSO in Our CASCompCert

Source $P$

- Clight

Target $P$

- CompCert
- x86
- x86-SC

Target $P_{TSO}$

- x86-TSO
- Lock-Impl

Call

Lock

Race-free

identity

Manually impl.

x86-TSO semantics

with benign-races
Supporting Confined Benign-Race & x86-TSO in Our CASCompCert

Source P

CompCert

Call

Lock

Race-free

identity

Target P

x86

CLight

Call

Lock

x86-SC

Manually impl.

Target P'_{TSO}

x86-TSO

Call

Lock-Impl

x86-TSO semantics with benign-races
Supporting Confined Benign-Race & x86-TSO in Our CASCompCert

Source $P$

CompCert

Target $P$

x86

Target $P'_{TSO}$

x86-TSO

Clight

Call

Lock

DRF

Race-free

.identity

Manually impl.

Lock-Impl

$\subseteq$

x86-TSO semantics

with benign-races

并向 invasive TSO semantics
Supporting Confined Benign-Race & x86-TSO in Our CASCompCert

Source $P$

$P$ \Rightarrow \text{CompCert} \Rightarrow \text{Clight} \Rightarrow \text{Lock} \Rightarrow \text{DRF}$

Target $P$

$P \Rightarrow \text{x86} \Rightarrow \text{identity} \Rightarrow \text{Lock} \Rightarrow \text{Race-free}$

$P'_{TSO}$

$P'_{TSO} \Rightarrow \text{x86-TSO} \Rightarrow \text{Lock-Impl} \Rightarrow \text{with benign-races}$
Supporting Confined Benign-Race & x86-TSO in Our CASCompCert

Source $P$

$\text{CompCert}$

$\text{Clight}$

$\implies$ $\text{Call}$

$\implies$ $\text{Lock}$

$\implies$ $\text{Race-free}$

$\implies$ $\text{DRF}$

Target $P$

$\text{x86}$

$\text{idemtity}$

$\implies$ $\text{Call}$

$\implies$ $\text{Lock}$

$\implies$ $\text{x86-SC}$

$\implies$ $\text{Call}$

Target $P'_{TSO}$

$\text{x86-TSO}$

$\text{Lock-Impl}$

$x86$-TSO semantics

with benign-races
Supporting Confined Benign-Race & x86-TSO in Our CASCompCert

Source $P$

UI

Target $P$

UI

Target $P'_{TSO}$

Clight

CompCert

x86

identity

x86-SC

x86-TSO semantics with benign-races

Call

Lock

Lock-Impl

DRF

Race-free

Call

Lock

identity

≤
Supporting Confined Benign-Race & x86-TSO in Our CASCompCert

Source $P$

$\text{UI}$

Target $P$

$\text{UI}$

Target $P_{TSO}$

Clight $\rightarrow$ CompCert $\rightarrow$ x86 $\rightarrow$ x86-TSO $\rightarrow$ Lock-Impl

Call $\rightarrow$ Lock $\rightarrow$ DRF

Lock $\rightarrow$ x86-SC

x86-TSO semantics with benign-races
Supporting Confined Benign-Race & x86-TSO in Our CASCompCert

Source \( \mathcal{P} \)

Target \( \mathcal{P} \)

Target \( \mathcal{P}_{TSO} \)

Clight

CompCert

x86

x86-TSO

Lock-Impl

x86-SC

Race-free

Identity

DRF

Call

Lock

Lock

Call

Call

x86-TSO semantics

with benign-races

\( \leq \)
Verified 12/20 Passes in CASCompCert

Verified 12/20 passes

Tailcall, Renumber, Inlining, Constprop, CSE, Deadcode, Unusedglob

20 passes
Verified 12/20 Passes in CASCompCert

Including all the translation passes from Clight to x86

Verified 12/20 passes
**Reused Considerable Amount of CompCert Proofs. Framework is Challenging to Implement.**

<table>
<thead>
<tr>
<th>Compilation passes</th>
<th>Spec CompCert</th>
<th>Ours</th>
<th>Proof CompCert</th>
<th>Ours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cshmgen</td>
<td>515</td>
<td>1021</td>
<td>1071</td>
<td>1503</td>
</tr>
<tr>
<td>Cminorgen</td>
<td>753</td>
<td>1556</td>
<td>1152</td>
<td>1251</td>
</tr>
<tr>
<td>Selection</td>
<td>336</td>
<td>500</td>
<td>647</td>
<td>783</td>
</tr>
<tr>
<td>RTLgen</td>
<td>428</td>
<td>543</td>
<td>821</td>
<td>862</td>
</tr>
<tr>
<td>Tailcall</td>
<td>173</td>
<td>328</td>
<td>275</td>
<td>405</td>
</tr>
<tr>
<td>Renumber</td>
<td>86</td>
<td>245</td>
<td>117</td>
<td>358</td>
</tr>
<tr>
<td>Allocation</td>
<td>704</td>
<td>785</td>
<td>1410</td>
<td>1700</td>
</tr>
<tr>
<td>Tunneling</td>
<td>131</td>
<td>339</td>
<td>166</td>
<td>475</td>
</tr>
<tr>
<td>Linearize</td>
<td>236</td>
<td>371</td>
<td>349</td>
<td>733</td>
</tr>
<tr>
<td>CleanupLabels</td>
<td>126</td>
<td>387</td>
<td>161</td>
<td>388</td>
</tr>
<tr>
<td>Stacking</td>
<td>730</td>
<td>1038</td>
<td>1108</td>
<td>2135</td>
</tr>
<tr>
<td>Asmgen</td>
<td>208</td>
<td>338</td>
<td>571</td>
<td>1128</td>
</tr>
</tbody>
</table>
We leave as future work.

In Coq we have mechanized the framework (Fig. 7.4 Proof Eguarantee of 

Let source object code (we use the object code does not preserve termination for now, which omission here). This is because our simulation Sec. 3.2.

As explained before, Lem. (see tl)

Theorem 1540

Compiler verification: 100 - 400 more lines of Coq proof for most passes

<table>
<thead>
<tr>
<th>Compilation passes</th>
<th>Spec</th>
<th>Proof</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CompCert</td>
<td>Ours</td>
</tr>
<tr>
<td>Cshmgen</td>
<td>515</td>
<td>1021</td>
</tr>
<tr>
<td>Cminorgen</td>
<td>753</td>
<td>1556</td>
</tr>
<tr>
<td>Selection</td>
<td>336</td>
<td>500</td>
</tr>
<tr>
<td>RTLgen</td>
<td>428</td>
<td>543</td>
</tr>
<tr>
<td>Tailcall</td>
<td>173</td>
<td>328</td>
</tr>
<tr>
<td>Renumber</td>
<td>86</td>
<td>245</td>
</tr>
<tr>
<td>Allocation</td>
<td>704</td>
<td>785</td>
</tr>
<tr>
<td>Tunneling</td>
<td>131</td>
<td>339</td>
</tr>
<tr>
<td>Linearize</td>
<td>236</td>
<td>371</td>
</tr>
<tr>
<td>CleanupLabels</td>
<td>126</td>
<td>387</td>
</tr>
<tr>
<td>Stacking</td>
<td>730</td>
<td>1038</td>
</tr>
<tr>
<td>Asmgen</td>
<td>208</td>
<td>338</td>
</tr>
</tbody>
</table>
**Reused Considerable Amount of CompCert Proofs. Framework is Challenging to Implement.**

<table>
<thead>
<tr>
<th>Compilation passes</th>
<th>Spec</th>
<th>Proof</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CompCert</td>
<td>Ours</td>
</tr>
<tr>
<td>Cshmgcn</td>
<td>515</td>
<td>1021</td>
</tr>
</tbody>
</table>

**Compiler verification:**
100 - 400 more lines of Coq proof for most passes
Reused Considerable Amount of CompCert Proofs. Framework is Challenging to Implement.

<table>
<thead>
<tr>
<th>Compilation passes</th>
<th>Spec</th>
<th>Proof</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CompCert</td>
<td>Ours</td>
</tr>
<tr>
<td>Cshmgen</td>
<td>515</td>
<td>1021</td>
</tr>
</tbody>
</table>

Compiler verification: 100 - 400 more lines of Coq proof for most passes

Framework impl.: > 60k LoC, ~ 1 person year
Conclusion
Conclusion

• **Language independent** verification framework
  - Key semantics components + proof structures
  - Supports separate compilation for *race-free* concurrent programs
  - **Well-defined language** for language-independent DRF
  - **Footprint-preserving simulation** for DRF preservation
Conclusion

• **Language independent** verification framework
  - Key semantics components + proof structures
  - Supports separate compilation for *race-free* concurrent programs
  - **Well-defined language** for language-independent DRF
  - **Footprint-preserving simulation** for DRF preservation

• Framework extension:
  - Support **x86-TSO** + **confined benign-races**
Conclusion

- **Language independent** verification framework
  - Key semantics components + proof structures
  - Supports separate compilation for **race-free** concurrent programs
  - **Well-defined language** for language-independent DRF
  - **Footprint-preserving simulation** for DRF preservation

- Framework extension:
  - Support **x86-TSO + confined benign-races**

- **CAS** CompCert:
  - Reused considerable amount of CompCert proofs
  - Racy x86-TSO impl. of locks as synchronization library
Thank you!