Atomicity Refinement for Verified Compilation

David Pichardie **Vincent Laporte** Gustavo Petri Jan Vitek
Suresh Jagannathan

INRIA, Purdue University, Université Rennes 1

January 25th, 2013

Context: Verified Compilation

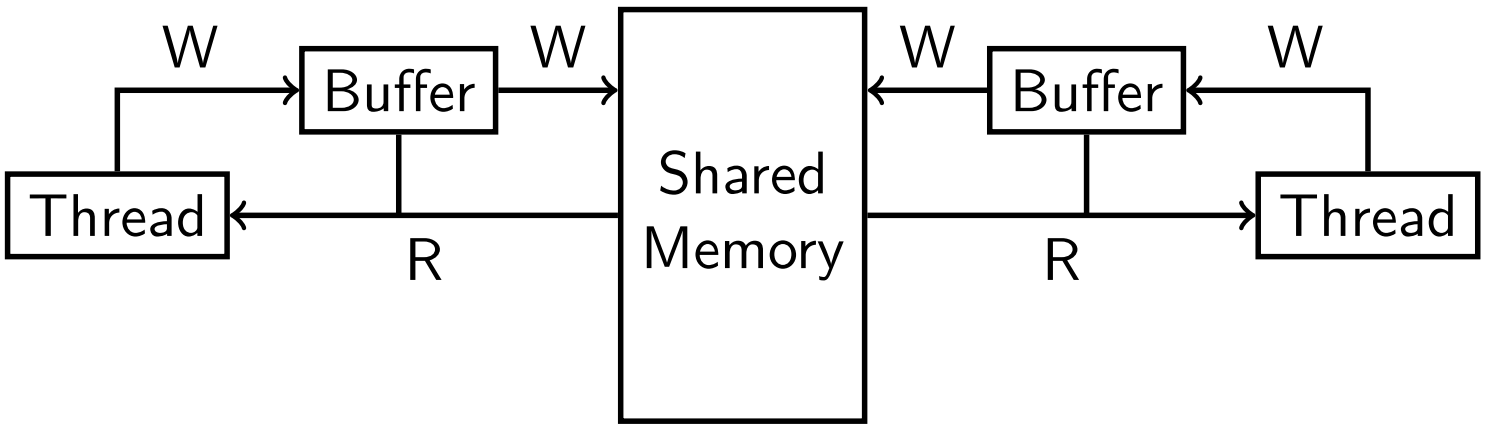
CompCert (Leroy 2006)

- Optimizing compiler from C to assembly
- Fully verified in the Coq proof assistant

Context: Verified Compilation

CompCert-TSO (Ševčík et al. 2011)

- TSO semantics for all intermediate languages from x86 to C



- low-level concurrency primitives: threads, compare-and-swap

This Work

Towards higher-level languages

general synchronization mechanisms

garbage collection

Strong formal guarantees

mechanized proofs

Methodology

Refinement: replace the implementation by a high-level specification

Running Example: Lock

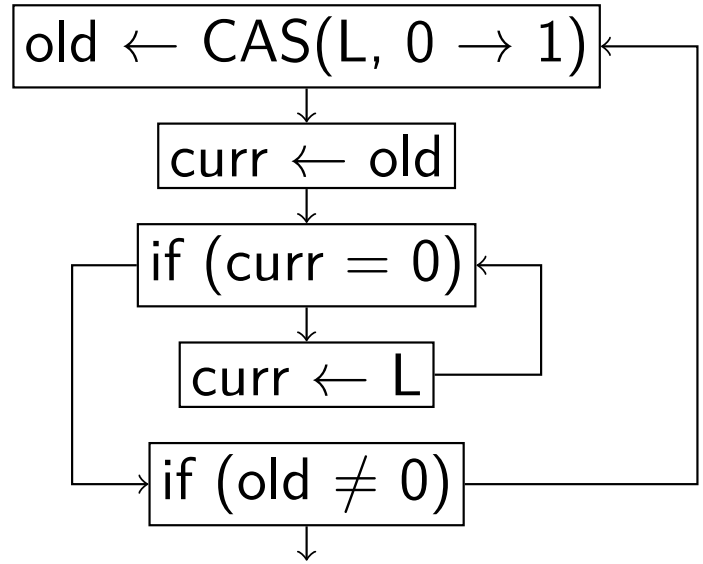
Lock L
  critical section
Unlock L

Running Example: Lock

Lock L

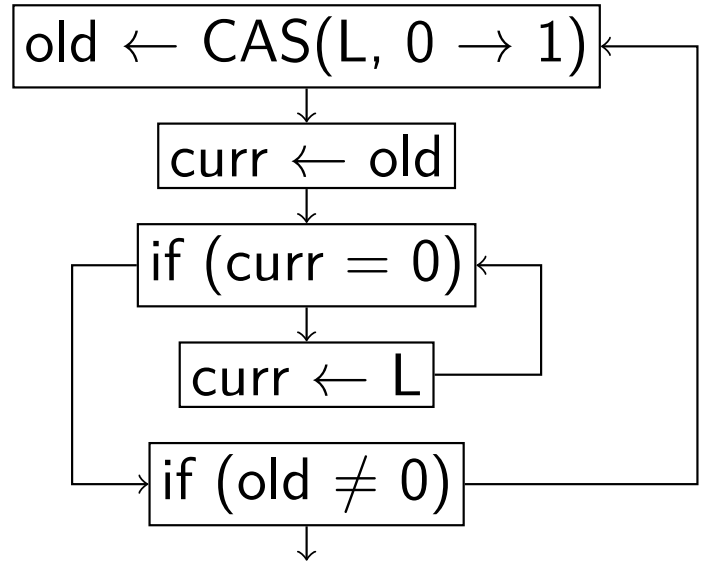
critical section

Unlock L



Running Example: Lock

Lock L
  critical section
Unlock L



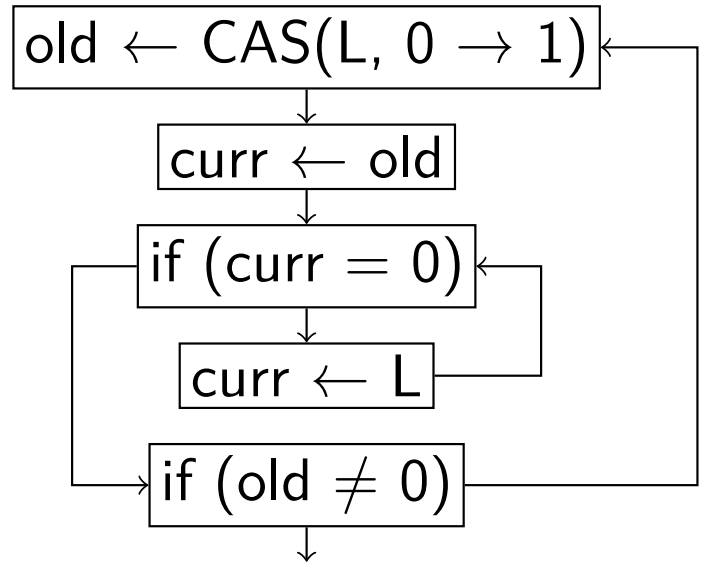
Proof technique: backward simulation

What? Prove that any trace of the target is a trace of the source.

How? Match every state in the target trace with a state in the source.

Running Example: Lock

Lock L
  critical section
Unlock L



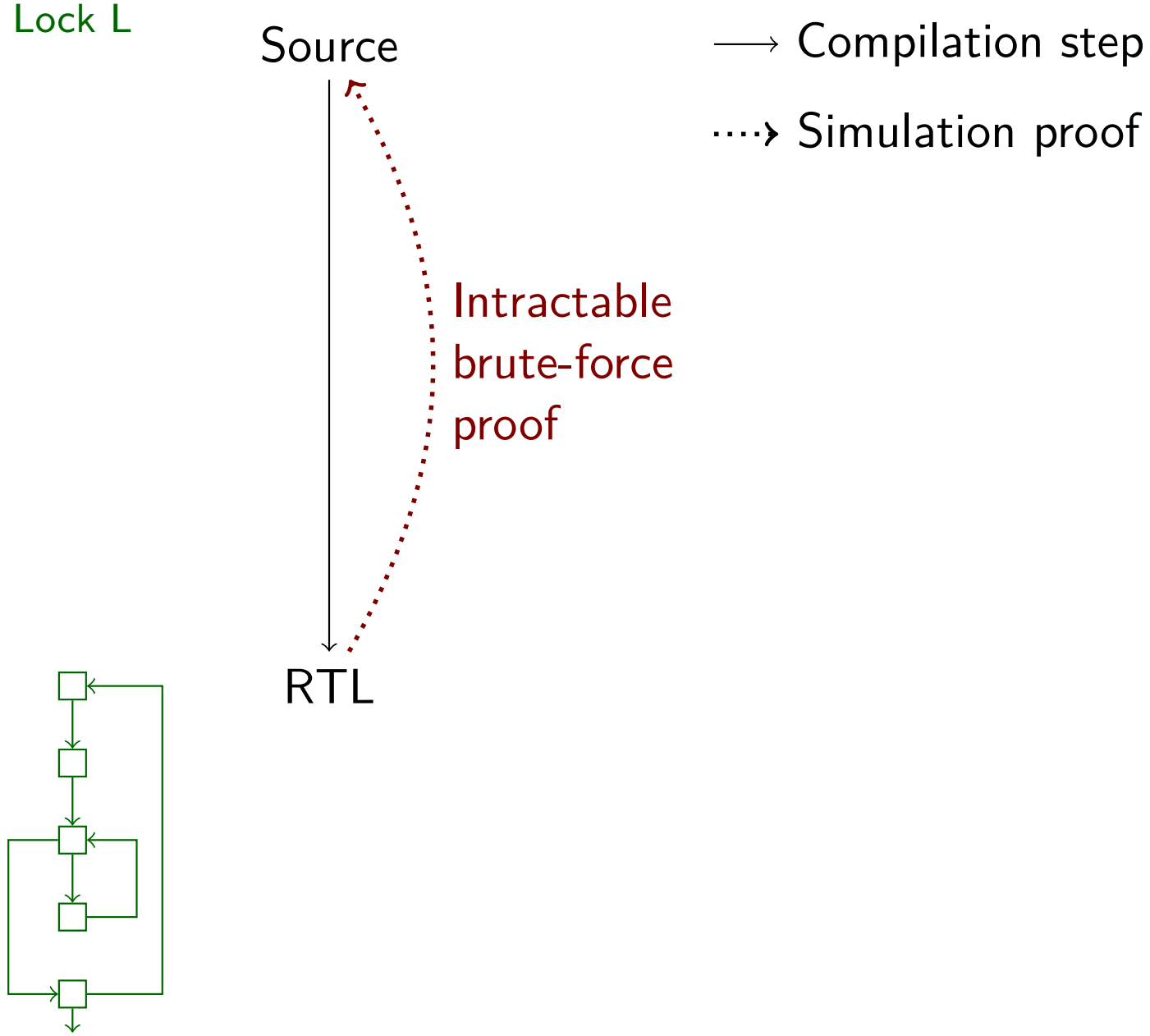
Proof technique: backward simulation

What? Prove that any trace of the target is a trace of the source.

How? Match every state in the target trace with a state in the source.

→ Too hard because one has to consider too many interleavings (and relaxed behaviors): compilation introduces memory accesses and loops.

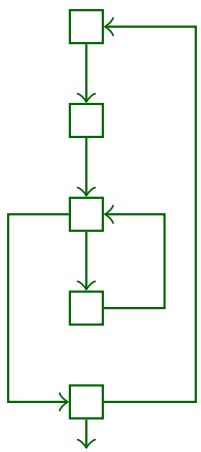
Compiler Architecture

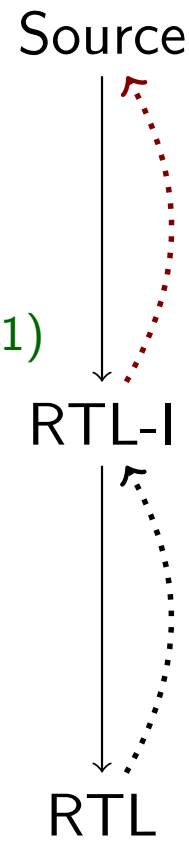


Compiler Architecture

Lock L

Repeat
  old $\leftarrow$ CAS(L, 0 $\rightarrow$ 1)
  curr $\leftarrow$ old
  While curr $= 0$
    curr $\leftarrow$ L
Until old $= 0$





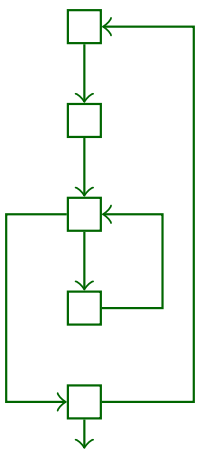
$\longrightarrow$ Compilation step

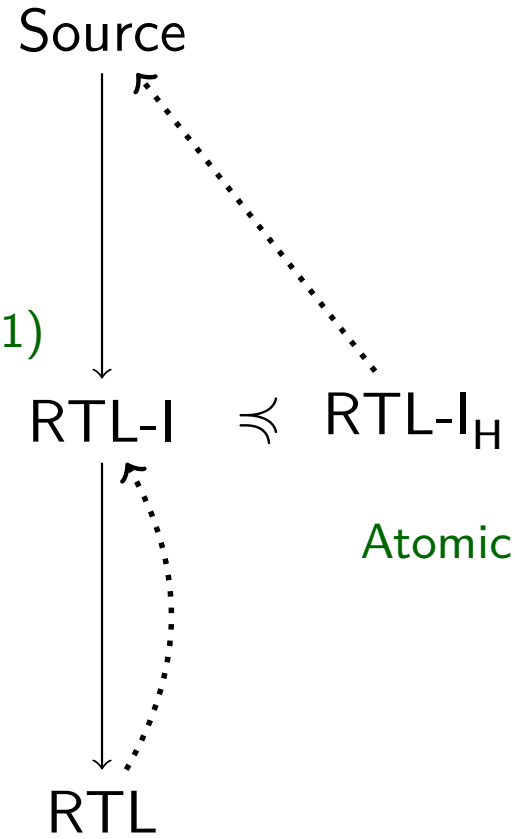
$\dashrightarrow$ Simulation proof

Compiler Architecture

Lock L

Repeat
  old ← CAS(L, 0 → 1)
  curr ← old
  While curr = 0
    curr ← L
Until old = 0





Atomic{ Assume(L = 0); L ← 1 }

⟶ Compilation step

⋯› Simulation proof

≼ Refinement

Refinement Rules

$$\textsc{Trans}\ \frac{s_1 \preccurlyeq s_2 \quad s_2 \preccurlyeq s_3}{s_1 \preccurlyeq s_3} \qquad \textsc{Repeat}\ \frac{}{\mathsf{repeat}\ s\ \mathsf{until}\ c \preccurlyeq \mathsf{loop}\ (s; \mathsf{assume}(\neg c)); s; \mathsf{assume}(c)} \qquad \textsc{IfBranch}\ \frac{}{\mathsf{if}\ \ \mathsf{then}\ s_1\ \mathsf{else}\ s_2 \preccurlyeq \mathsf{branch}\ (\mathsf{assume}(c); s_1),\ (\mathsf{assume}(\neg c); s_2)}$$

$$\textsc{IfAtomic}\ \frac{}{\mathsf{if}\ c\ \mathsf{then}\ (\mathsf{atomic}\ s_0)\ \mathsf{else}\ (\mathsf{atomic}\ s_1) \preccurlyeq \mathsf{atomic}\ (\mathsf{if}\ c\ \mathsf{then}\ s_0\ \mathsf{else}\ s_1)} \qquad \textsc{EFLeft}\ \frac{s_1\ \text{is effect free}}{s_1; \mathsf{atomic}\ s_2 \preccurlyeq \mathsf{atomic}\ (s_1; s_2)} \qquad \textsc{EFRight}\ \frac{s_2\ \text{is effect free}}{\mathsf{atomic}\ s_1; s_2 \preccurlyeq \mathsf{atomic}\ (s_1; s_2)}$$

$$\textsc{CAS-fail}\ \frac{}{\mathsf{cas}(d, r, o, n); \mathsf{assume}(o \neq d) \preccurlyeq \mathsf{fence}; \mathsf{load}\,(d, r)} \qquad \textsc{CAS-success}\ \frac{}{\mathsf{cas}(d, r, o, n); \mathsf{assume}(o = d) \preccurlyeq \mathsf{atomic}\ (\mathsf{load}\,(d, r); \mathsf{assume}(o = d); \mathsf{store}\,(n, r))}$$

$$\textsc{SwapAssume}\ \frac{\mathsf{defines}(s) \cap \mathsf{uses}(c) = \emptyset}{s; \mathsf{assume}(c) \preccurlyeq \mathsf{assume}(c); s} \qquad \textsc{Dead}\ \frac{s_1\ \text{is dead}}{s_1; s_2 \preccurlyeq s_2} \qquad \textsc{FenceAtomic}\ \frac{}{\mathsf{fence} \preccurlyeq \mathsf{atomic}\,\mathsf{skip}} \qquad \textsc{FenceElim}\ \frac{}{\mathsf{fence} \preccurlyeq \mathsf{skip}} \qquad \textsc{AfterAbort}\ \frac{}{\mathsf{abort}; s \preccurlyeq \mathsf{abort}}$$

$$\textsc{GrowAtomicLocal}\ \frac{s_0 \in \{\mathsf{load}_{\mathsf{Loc}}(r, d), \mathsf{store}_{\mathsf{Loc}}(d, r)\}}{s_0; \mathsf{atomic}\ s_1 \preccurlyeq \mathsf{atomic}\ (s_0; s_1)} \qquad \textsc{MakeStoreAtomic}\ \frac{s \in \{\mathsf{store}(r, d), \mathsf{release\ store}\,(r, d)\}}{s; \mathsf{fence} \preccurlyeq \mathsf{atomic}\ s} \qquad \textsc{MakeLoadAtomic}\ \frac{}{\mathsf{fence}; \mathsf{load}\,(r, d) \preccurlyeq \mathsf{atomic}\,\mathsf{load}\,(r, d)}$$

Interactive Refinement Proof

```
Repeat
  old ← CAS(L, 0 → 1)
  curr ← old
  While curr = 0
    curr ← L
Until old = 0
```

$\preccurlyeq$

```
Loop
  old ← CAS(L, 0 → 1)
  curr ← old
  While curr = 0
    curr ← L
  Assume(old ≠ 0)
old ← CAS(L, 0 → 1)
curr ← old
While curr = 0
  curr ← L
Assume(old = 0)
```

$$\frac{}{\text{Repeat s Until c} \preccurlyeq \text{Loop } \{\text{ s; Assume}(\neg\text{c}) \}\text{; s; Assume(c)}}$$

Interactive Refinement Proof

```
Loop
  old ← CAS(L, 0 → 1)
  curr ← old
  While curr = 0
    curr ← L
  Assume(old ≠ 0)
old ← CAS(L, 0 → 1)
curr ← old
While curr = 0
  curr ← L
Assume(old = 0)
```

$\preccurlyeq$

```
Loop
  old ← CAS(L, 0 → 1)
  Assume(old ≠ 0)
  curr ← old
  While curr = 0
    curr ← L
old ← CAS(L, 0 → 1)
Assume(old = 0)
curr ← old
While curr = 0
  curr ← L
```

$$\frac{\mathsf{defines}(s) \cap \mathsf{uses}(c) = \emptyset}{s;\ \mathsf{Assume}(c) \preccurlyeq \mathsf{Assume}(c);\ s}$$

Interactive Refinement Proof

Loop
  old $\leftarrow$ CAS(L, 0 $\rightarrow$ 1)
  Assume(old $\neq$ 0)
  curr $\leftarrow$ old
  While curr = 0
    curr $\leftarrow$ L
old $\leftarrow$ CAS(L, 0 $\rightarrow$ 1)
Assume(old = 0)
curr $\leftarrow$ old
While curr = 0
  curr $\leftarrow$ L

$\preccurlyeq$

Loop
  Fence
  old $\leftarrow$ L
  curr $\leftarrow$ old
  While curr = 0
    curr $\leftarrow$ L
old $\leftarrow$ CAS(L, 0 $\rightarrow$ 1)
Assume(old = 0)
curr $\leftarrow$ old
While curr = 0
  curr $\leftarrow$ L

---

o $\leftarrow$ CAS(d,e $\rightarrow$ n); Assume(o $\neq$ e) $\preccurlyeq$ Fence; o $\leftarrow$ d

Interactive Refinement Proof

```
Loop                                  Loop
  Fence                                 Fence
  old ← L                               old ← L
  curr ← old                            curr ← old
  While curr = 0                        While curr = 0
    curr ← L                              curr ← L
old ← CAS(L, 0 → 1)                   Atomic
Assume(old = 0)                         old ← L
curr ← old                              Assume(old = 0)
While curr = 0                          L ← 1
  curr ← L                            curr ← old
                                      While curr = 0
                                        curr ← L
```

$\preccurlyeq$

$o \leftarrow CAS(d,e \rightarrow n); Assume(o = e) \preccurlyeq$

$Atomic \{ o \leftarrow d; Assume(o = e); d \leftarrow n \}$

Interactive Refinement Proof

Loop
  Fence
  old $\leftarrow$ L
  curr $\leftarrow$ old
  While curr $= 0$
    curr $\leftarrow$ L
Atomic
  old $\leftarrow$ L
  Assume(old $= 0$)
  L $\leftarrow 1$
curr $\leftarrow$ old
While curr $= 0$
  curr $\leftarrow$ L

$\preccurlyeq$

Atomic
  old $\leftarrow$ L
  Assume(old $= 0$)
  L $\leftarrow 1$

$$\frac{s_1 \text{ is dead}}{s_1; s_2 \preccurlyeq s_2}$$

Summary

- Compiler for a high-level concurrent language written in Coq
- Proof that program refinement implies behaviour inclusion
- Sound refinement rules
- Case studies: spin lock and concurrent garbage collector (using a permission system to handle such complex examples)

Summary

- Compiler for a high-level concurrent language written in Coq
- Proof that program refinement implies behaviour inclusion
- Sound refinement rules
- Case studies: spin lock and concurrent garbage collector (using a permission system to handle such complex examples)

Questions?