Formally verified semantic equivalence automatic checker for x86 and CHERI

Swarn Priya, Freek Verbeek, Binoy Ravindran

May 21, 2025

1 Problem Statement

Proving semantic equivalence between x86 and CHERI (Capability Hardware Enhanced RISC Instructions) programs is a critical step toward enabling the secure and verifiable migration of legacy software to capability-based architectures. This challenge arises from the need to reconcile two fundamentally different models of memory and security: while x86 relies on untyped, pointer-based memory access, CHERI introduces hardware-enforced capabilities that provide fine-grained spatial and temporal memory safety. Demonstrating equivalence entails constructing a formal correspondence between the two systems' low-level memory operations, control flow semantics, and calling conventions. Such a mapping ensures that for every observable behavior in an x86 program, the corresponding CHERI version produces the same outcome while adhering to the strict safety guarantees of the CHERI model.

2 Solution

Build a formally verified checker in Coq based on "Abstract Interpretation". The role of the checker is to perform static analysis on x86 and CHERI (ARM/RISC-V) to prove semantic equivalence between them. Semantic equivalence between x86 and CHERI programs refers to the property that both programs, despite being written or compiled for architectures with different execution and memory models, exhibit the same(observable) behavior when executed on their respective platforms. This equivalence requires a formal relation between:

- Memory models – x86 uses flat, untyped memory accessed via raw pointers; CHERI uses capabilities that encode bounds and permissions.
- Register maps - x86 and CHERI has different set of registers
- Pointer semantics – CHERI capabilities must be shown to faithfully implement the same addressing and dereferencing behavior as raw x86 pointers, within safety constraints.

- Control flow and calling conventions – Function calls, returns, and jumps must preserve the same logic and reachability across both programs.
- Side effects – Any observable effects (e.g., writes to files, network packets sent, system calls invoked) must match.

2.1 Methodology

Concrete World:

- Concrete World: Semantic equivalence: (Only valid for programs with defined behavior)

$$\forall P_{x86}, s_x, s'_x, P_c, s_c, s'_c, \quad P_{x86} : s_x \to s'_x \quad \wedge \quad P_c : s_c \to_c s'_c \quad \wedge \quad s_x \sim s_c \implies s'_x \sim s'_c$$

$$\forall P_{x86}, s_x, P_c, s_c, \quad P_{x86} : s_x \uparrow \ \wedge s_x \sim s_c \implies \quad P_c : s_c \uparrow$$

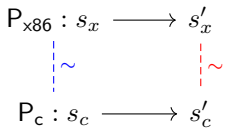

Figure 1: Concrete World

- **Simulation relation $\sim$ in Concrete World**:

  $\mathsf{x86_{st}}$: $\{\mathsf{xm} : \mathsf{x86_{mem}}; \mathsf{xp} : \mathsf{list\ xasm}; \mathsf{xip} : \mathsf{nat}\}$, where
  $\mathsf{x86_{mem}}$: $\{\mathsf{xmem} : \mathsf{x86_{mmap}}; \mathsf{xreg} : \mathsf{x86_{rmap}}; \ldots\}$, where
  $\mathsf{x86_{mmap}}$ is a mapping from addresses to values and $\mathsf{x86_{rmap}}$ is a mapping from registers to values.

  $\mathsf{CHERI_{st}}$: $\{\mathsf{cm} : \mathsf{CHERI_{mem}}; \mathsf{cp} : \mathsf{list\ casm}; \mathsf{cip} : \mathsf{nat}\}$, where
  $\mathsf{CHERI_{mem}}$: $\{\mathsf{cmem} : \mathsf{CHERI_{mmap}}; \mathsf{creg} : \mathsf{CHERI_{rmap}}; \ldots\}$, where
  $\mathsf{CHERI_{mmap}}$ is a mapping from addresses to values and $\mathsf{CHERI_{rmap}}$ is a mapping from registers to values.

  $\mathsf{x86_{st}} \sim \mathsf{CHERI_{st}}$:

  - Given a mapping $\Gamma : pc * \mathsf{x86_{reg}} \to \mathsf{CHERI_{reg}}$, we can define the $\sim$ relation between $\mathsf{x86_{st}}$ and $\mathsf{CHERI_{st}}$ as:

    - Γ: The mapping associates x86 registers—captured at each program point/basic block—with their corresponding CHERI registers, taking into account capability semantics and architectural differences. Essentially, it specifies how, in a given program, a set of x86 registers is translated to a set of CHERI registers to preserve the intended behavior. This mapping is inherently program-specific and cannot be fixed in advance; it must be generated dynamically as part of the translation process from x86 to CHERI. Therefore, the toolchain responsible for this transformation must also produce the register mapping, as it is a crucial component for defining the simulation relation $\sim$. Importantly, we only need to prove equivalence for those registers and memory addresses that are included in this mapping. We need not worry about extra capabilities registers that are being set on CHERI side.

Abstract World:

- Abstract World: Our goal is to perform static analysis to determine the semantic equivalence between two programs. To achieve this, we aim to leverage techniques from abstract interpretation. Rather than constructing a semantic equivalence proof manually for each program pair, we seek to design an automated checker that compares the abstract states of the two programs and determines whether they are equivalent. This approach enables systematic reasoning about program equivalence through abstraction.
  - **Abstraction function**: $\alpha(\mathsf{x86}_{\mathsf{st}}) = \mathsf{x86}_{\mathsf{ast}}$
  - **Abstraction function**: $\alpha(\mathsf{CHERI}_{\mathsf{st}}) = \mathsf{CHERI}_{\mathsf{ast}}$
  - Point to note: We aim to define a unified abstract state representation that can capture both x86 and CHERI concrete states within a single framework. One possible approach is to map concrete registers from each architecture to a common numerical identifier—for instance, mapping RAX, RCX, etc., in x86 to numbers like 1, 3, etc., using an abstraction function alpha, and similarly assigning identifiers to CHERI registers. The values stored in these registers can then be modeled using a shared abstract domain, such as intervals. This unified abstraction enables us to represent both architectures within a single abstract interpreter, avoiding the complexity of developing and maintaining two separate analyses.
  - We prove "Galois Connection" for both x86 and CHERI.
    - Define abstract states (One common for both x86 and CHERI)
    - Define abstract semantics for x86 and CHERI
    - Prove abstract semantics over approximates concrete semantics

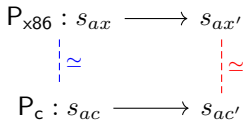

Figure 2: Abstract World

- **Simulation relation $\simeq$ in Abstract World**:
  Should be defined in similar way like we define for concrete world, but should be parameterized over abstract states (abstract mem, abstract regmap, ...)

  We want to prove simulation relation for abstract semantics because we want to show that the result given by our checker is correct:

$$\begin{aligned} &\forall P_{x86}, s_{ax}, s_{ax'}, P_c, s_{ac}, s_{ac'}, \\ &\texttt{CHECKER}(\mathsf{s_{ax}}, \mathsf{x86}, \mathsf{s_{ac}}, \texttt{CHERI}) = \texttt{YES} \quad \wedge \\ s_{ax} \simeq s_{ac} \quad \wedge \quad P : s_{ax} \to s_{ax'} \quad \wedge \quad &P_c : s_{ac} \to_c s_{ac'} \\ &\implies s_{ax'} \simeq s_{ac'} \end{aligned}$$

  Since, we prove seperately that our abstract semantics overapproximates the concrete semantics, this theorem will imply that the programs are equivalent.

  Or, we can prove:

$$\begin{aligned} &\forall P_{x86}, s_{ax}, s_{ax'}, P_c, s_{ac}, \\ &\mathsf{CHECKER}(\mathsf{s_{ax}}, \mathsf{x86}, \mathsf{s_{ac}}, \mathsf{CHERI}) = YES \quad \wedge \\ s_{ax} \simeq s_{ac} \quad \wedge \quad &P : s_{ax} \to s_{ax'} \implies \\ &\exists s_{ac'}, P_c : s_{ac} \to_c s_{ac'} \wedge s_{ax'} \simeq s_{ac'} \end{aligned}$$

  This lemma ensures that if the checker gives "yes" and we know that we have a defined semantics in x86 then for sure there exists a defined semantics in CHERI.

3 Prior Development to Reuse

- : Paper: SECOMP: Formally Secure Compilation of Compartmentalized C Programs
  - Coq development of CHERI RISC-V **SECOMP-Repo SECOMP-Paper**
- x-86 in Coq

  - CompCert's x86: **CompCert's x86**
  - Jasmin's x86: **Jasmin's x86**
  - Signedness Abstract Interpreter for while lang in Coq **AI-Coq**

4 Steps to explore

- As a first step, we will try to design the overall approach for programs just dealing with registers.
- Take the existing development of x86/CHERI and build abstract interpretation on top of it.
- Manually try to translate very small x86 program to CHERI and design the Γ function.