Synthesizing HW-SW Leakage Contracts for RISC-V Open-Source Processors

(to appear at DATE 2024)

Jan Reineke @ UNIVERSITÄT DES SAARLANDES

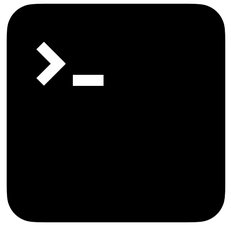
Joint work with
Gideon Mohr @ Universität des Saarlandes
Marco Guarnieri @ IMDEA Software

This work has received funding from an Intel Strategic Research Alliance (ISRA) and the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 101020415)"

The Need for New HW/SW Contracts

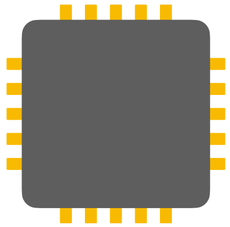
Instruction Set Architectures (ISAs): Benefits

High-level language



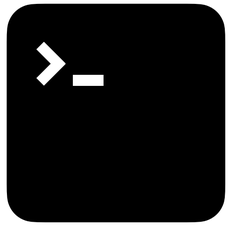
Instruction set architecture (ISA)

Microarchitecture



Instruction Set Architectures (ISAs): Benefits

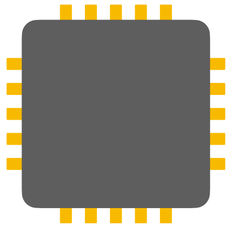
High-level language



Can program **independently** of microarchitecture

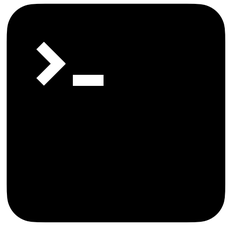
Instruction set architecture (ISA)

Microarchitecture



Instruction Set Architectures (ISAs): Benefits

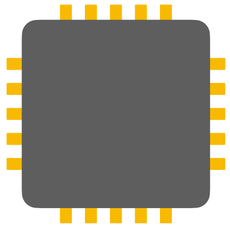
High-level language



Can program **independently** of microarchitecture

Instruction set architecture (ISA)

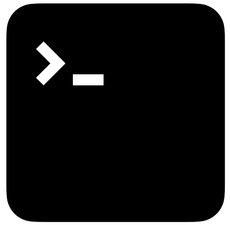
Microarchitecture



Can implement **arbitrary optimizations** as long as ISA semantics are obeyed

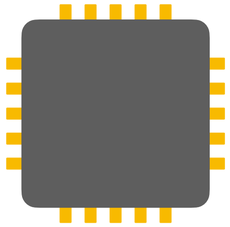
Inadequacy of ISAs: Side Channels

High-level language



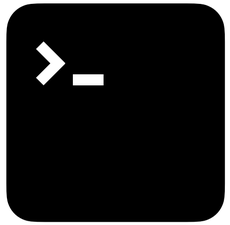
Instruction set architecture (ISA)

Microarchitecture



Inadequacy of ISAs: Side Channels

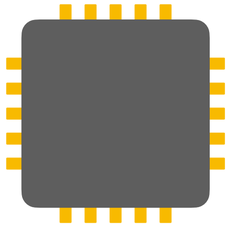
High-level language



Instruction set architecture (ISA)

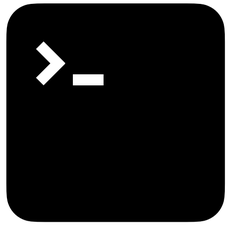
No guarantees about side channels

Microarchitecture



Inadequacy of ISAs: Side Channels

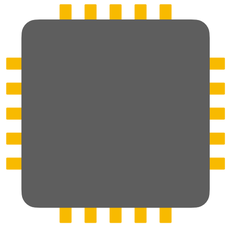
High-level language



Instruction set architecture (ISA)

No guarantees about side channels

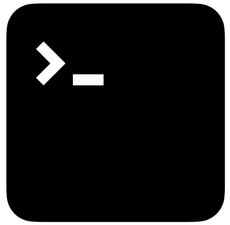
Microarchitecture



Can implement arbitrary **insecure** optimizations as long as ISA is implemented correctly

Inadequacy of ISAs: Side Channels

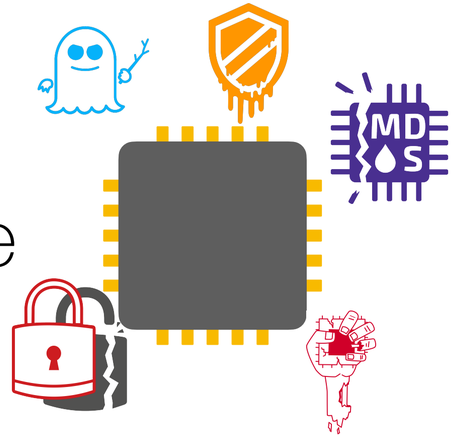
High-level language



Instruction set architecture (ISA)

No guarantees about side channels

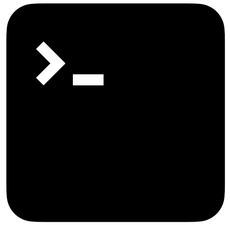
Microarchitecture



Can implement arbitrary **insecure** optimizations as long as ISA is implemented correctly

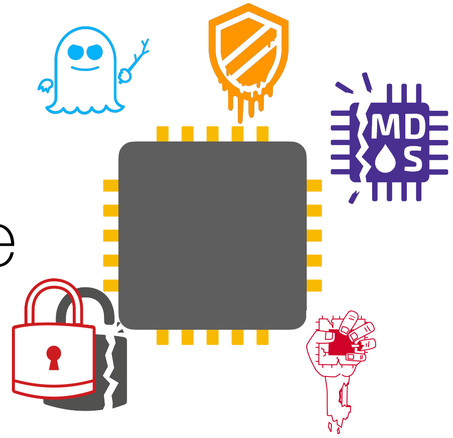
Inadequacy of ISAs: Side Channels

High-level language



Instruction set architecture (ISA)

Microarchitecture



Impossible to program securely
cryptographic algorithms?
sandboxing untrusted code?

No guarantees about side channels

Can implement arbitrary **insecure** optimizations as long as ISA is implemented correctly

A Way Forward: HW/SW Leakage Contracts

HW/SW Leakage contract =
ISA + Leakage specification

A Way Forward: HW/SW Leakage Contracts

Can program **securely** on top of contract
independently of microarchitecture

HW/SW Leakage contract =
ISA + Leakage specification

A Way Forward: HW/SW Leakage Contracts

Can program **securely** on top of contract
independently of microarchitecture

HW/SW Leakage contract =
ISA + Leakage specification

Can implement **arbitrary** ~~insecure~~ **optimizations**
as long as contract is obeyed

A Way Forward: HW/SW Leakage Contracts

Can program **securely** on top of contract **independently** of microarchitecture

HW/SW Leakage contract = ISA + Leakage specification

Captures possible leakage at ISA level

Can implement **arbitrary** ~~insecure~~ **optimizations** as long as contract is obeyed

Our prior work in this context

M. Guarnieri, B. Köpf, J. Reineke, and P. Vila
Hardware-Software Contracts for Secure Speculation
S&P (Oakland) 2021

Z. Wang, G. Mohr, K. v. Gleissenthall, J. Reineke, M. Guarnieri
Specification and Verification of Side-channel Security for Open-source Processors via Leakage Contracts
CCS (2023)

Synthesizing Leakage Contracts

Can program **securely** on top of contract **independently** of microarchitecture

HW/SW leakage contract = ISA + Leakage specification

Captures possible leakage at ISA level

Synthesizing Leakage Contracts

Can program **securely** on top of contract **independently** of microarchitecture

HW/SW leakage contract = ISA + Leakage specification

Captures possible leakage at ISA level

Can implement **arbitrary** ~~insecure~~ **optimizations** as long as contract is obeyed

Synthesizing Leakage Contracts

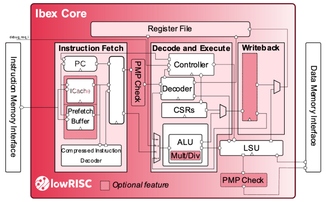
Can program **securely** on top of contract **independently** of microarchitecture

HW/SW leakage contract = ISA + Leakage specification

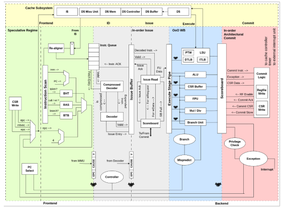
Captures possible leakage at ISA level

Can implement **arbitrary** ~~insecure~~ **optimizations** as long as contract is obeyed

RISC-V Open-Source Cores



Ibex



CVA6

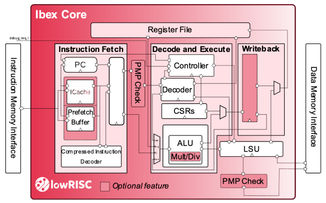
Synthesizing Leakage Contracts

Can program **securely** on top of contract **independently** of microarchitecture

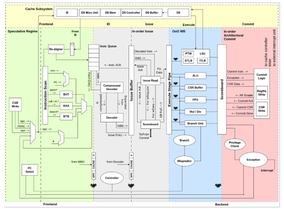
HW/SW leakage contract = ISA + Leakage specification

Captures possible leakage at ISA level

RISC-V Open-Source Cores



Ibex



CVA6

Synthesizing Leakage Contracts

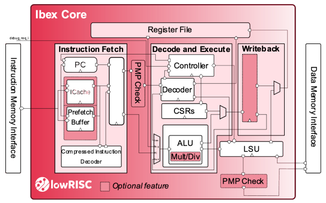
Can program **securely** on top of contract **independently** of microarchitecture

HW/SW leakage contract = ISA + Leakage specification

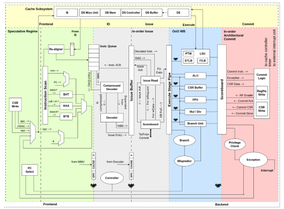
Captures possible leakage at ISA level

Contract Synthesis

RISC-V Open-Source Cores



Ibex



CVA6

Outline

1. Contracts and Contract Templates
2. Synthesis Goals and Methodology
3. Some Experimental Results

Outline

1. Contracts and Contract Templates

2. Synthesis Goals and Methodology

3. Some Experimental Results

An Example Contract

“For every multiplication the second operand leaks and
for every memory access the memory address leaks.”

An Example Contract

Composition of Contract Atoms

"For every multiplication the second operand leaks and for every memory access the memory address leaks."

An Example Contract

Composition of Contract Atoms

"For every multiplication the second operand leaks and for every memory access the memory address leaks."

Condition on ISA State

An Example Contract

Composition of Contract Atoms

"For every multiplication the second operand leaks and for every memory access the memory address leaks."

Condition on ISA State

Leakage of Part of ISA State

Contract Atoms -
What could possibly leak (non-speculatively)?

```
mul a0, t0, t1
```

$a0 \leftarrow t0 * t1$

Leakage sources:

Contract Atoms - What could possibly leak (non-speculatively)?

```
mul a0, t0, t1
```

Instruction type

```
a0 ← t0 * t1
```

Leakage sources:

- Instruction type

Contract Atoms - What could possibly leak (non-speculatively)?

```
mul a0, t0, t1
```

Register names and values

```
a0 ← t0 * t1
```

Leakage sources:

- Instruction type
- Register names
- Register values

Contract Atoms - What could possibly leak (non-speculatively)?

```
mul a0, t0, t1        a0 ← t0 * t1
```

Leakage sources:

- Instruction type
- Register names
- Register values

```
lw  a1, -4(t2)        a1 ← mem[t2 + (-4)]
```

Contract Atoms - What could possibly leak (non-speculatively)?

```
mul a0, t0, t1        a0 ← t0 * t1

lw  a1, -4(t2)        a1 ← mem[t2 + (-4)]
```

Immediate values

Leakage sources:

- Instruction type
- Register names
- Register values
- Immediate values

Contract Atoms - What could possibly leak (non-speculatively)?

```
mul a0, t0, t1        a0 ← t0 * t1

lw  a1, -4(t2)        a1 ← mem[t2 + (-4)]
```

Memory addresses and values

Leakage sources:

- Instruction type
- Register names
- Register values
- Immediate values
- Memory addresses
- Memory values

Contract Atoms - What could possibly leak (non-speculatively)?

```
mul a0, t0, t1        a0 ← t0 * t1

lw  a1, -4(t2)        a1 ← mem[t2 + (-4)]
```

Leakage sources:

- Instruction type
- Register names
- Register values
- Immediate values
- Memory addresses
- Memory values

Contract Atoms -
What could possibly leak (non-speculatively)?

Contract Atom =

Contract Atoms - What could possibly leak (non-speculatively)?

Contract Atom =

Condition based on Instruction Type

Contract Atoms - What could possibly leak (non-speculatively)?

Contract Atom =

Condition based on Instruction Type x Applicable Leakage Sources

Contract Atoms - What could possibly leak (non-speculatively)?

Contract Atom =

Condition based on Instruction Type x Applicable Leakage Sources

→ Several hundred contract atoms for RISC-V I+M

Outline

1. Contracts and Contract Templates

2. Synthesis Goals and Methodology

3. Some Experimental Results

Contract Synthesis: Goals

Contract Synthesis: Goals

For a given processor, find contract from template s.t.:

Contract Synthesis: Goals

For a given processor, find contract from template s.t.:

1. Processor satisfies contract

Contract Synthesis: Goals

For a given processor, find contract from template s.t.:

1. Processor satisfies contract

2. Contract is as precise as possible

Contract Synthesis: Goals

For a given processor, find contract from template s.t.:

1. Processor satisfies contract

For any pair of programs and inputs (P_1, I_1) and (P_2, I_2):

2. Contract is as precise as possible

Contract Synthesis: Goals

For a given processor, find contract from template s.t.:

1. Processor satisfies contract

For any pair of programs and inputs (P_1, I_1) and (P_2, I_2):

 $(P_1, I_1) \neq$  (P_2, I_2)

2. Contract is as precise as possible

Contract Synthesis: Goals

For a given processor, find contract from template s.t.:

1. Processor satisfies contract

For any pair of programs and inputs (P_1, I_1) and (P_2, I_2):

 $(P_1, I_1) \neq$  $(P_2, I_2) \implies$  $(P_1, I_1) \neq$  (P_2, I_2)

2. Contract is as precise as possible

Contract Synthesis: Goals

For a given processor, find contract from template s.t.:

1. Processor satisfies contract

For any pair of programs and inputs (P_1, I_1) and (P_2, I_2):

 $(P_1, I_1) \neq$  $(P_2, I_2) \implies$  $(P_1, I_1) \neq$  (P_2, I_2)

2. Contract is as precise as possible

As few programs and inputs (P, I) as possible s.t.:

Contract Synthesis: Goals

For a given processor, find contract from template s.t.:

1. Processor satisfies contract

For any pair of programs and inputs (P_1, I_1) and (P_2, I_2):

$$\text{🕵}(P_1, I_1) \neq \text{🕵}(P_2, I_2) \implies \text{📄}(P_1, I_1) \neq \text{📄}(P_2, I_2)$$

2. Contract is as precise as possible

As few programs and inputs (P, I) as possible s.t.:

$$\text{📄}(P_1, I_1) \neq \text{📄}(P_2, I_2)$$

Contract Synthesis: Goals

For a given processor, find contract from template s.t.:

1. Processor satisfies contract

For any pair of programs and inputs (P_1, I_1) and (P_2, I_2):

$$\text{🕵}(P_1, I_1) \neq \text{🕵}(P_2, I_2) \implies \text{📄}(P_1, I_1) \neq \text{📄}(P_2, I_2)$$

2. Contract is as precise as possible

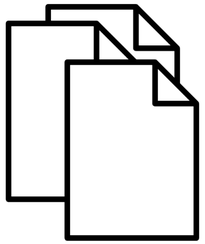
As few programs and inputs (P, I) as possible s.t.:

$$\text{📄}(P_1, I_1) \neq \text{📄}(P_2, I_2) \wedge \text{🕵}(P_1, I_1) = \text{🕵}(P_2, I_2)$$

Contract Synthesis: Methodology (1/2)

Contract Synthesis: Methodology (1/2)

Use test cases as proxy for contract satisfaction

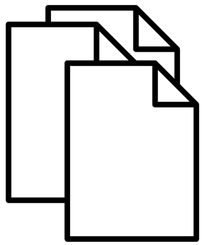


Test case =

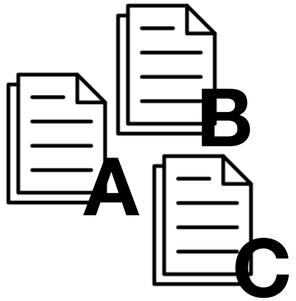
Pair (P_1, I_1), (P_2, I_2)

Contract Synthesis: Methodology (1/2)

Use test cases as proxy for contract satisfaction



Test case =
Pair (P_1, I_1), (P_2, I_2)

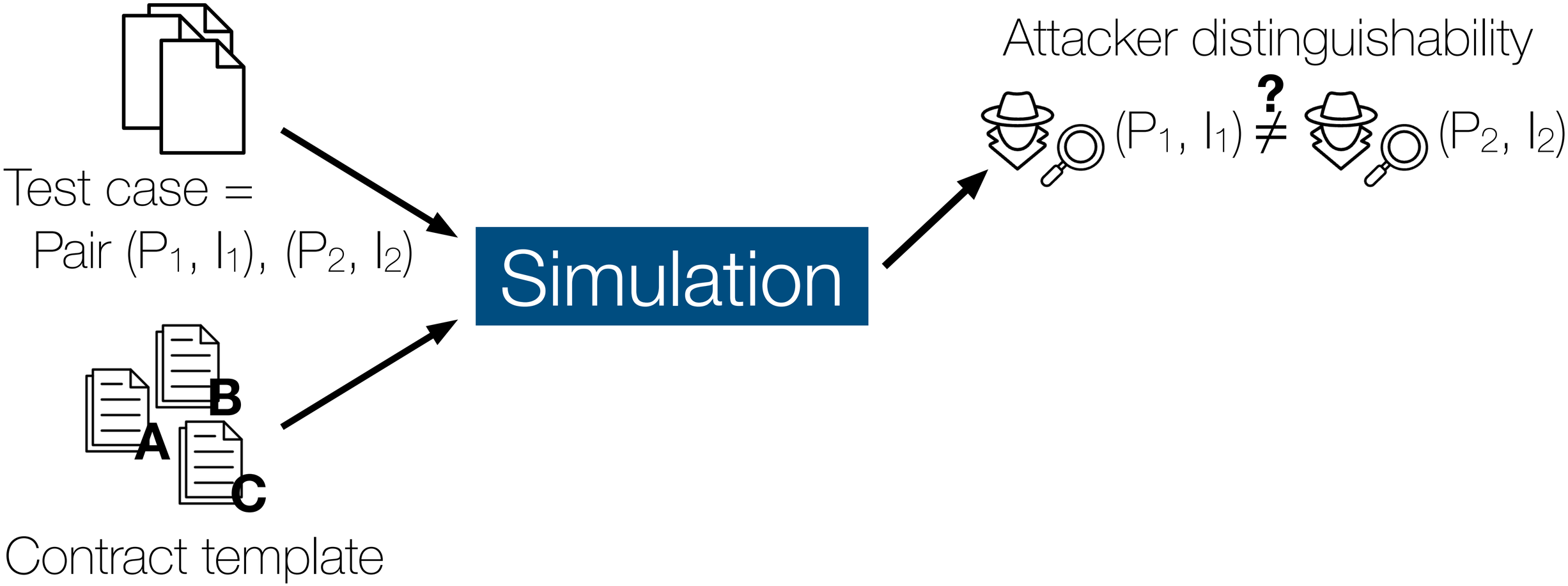


Contract template

Simulation

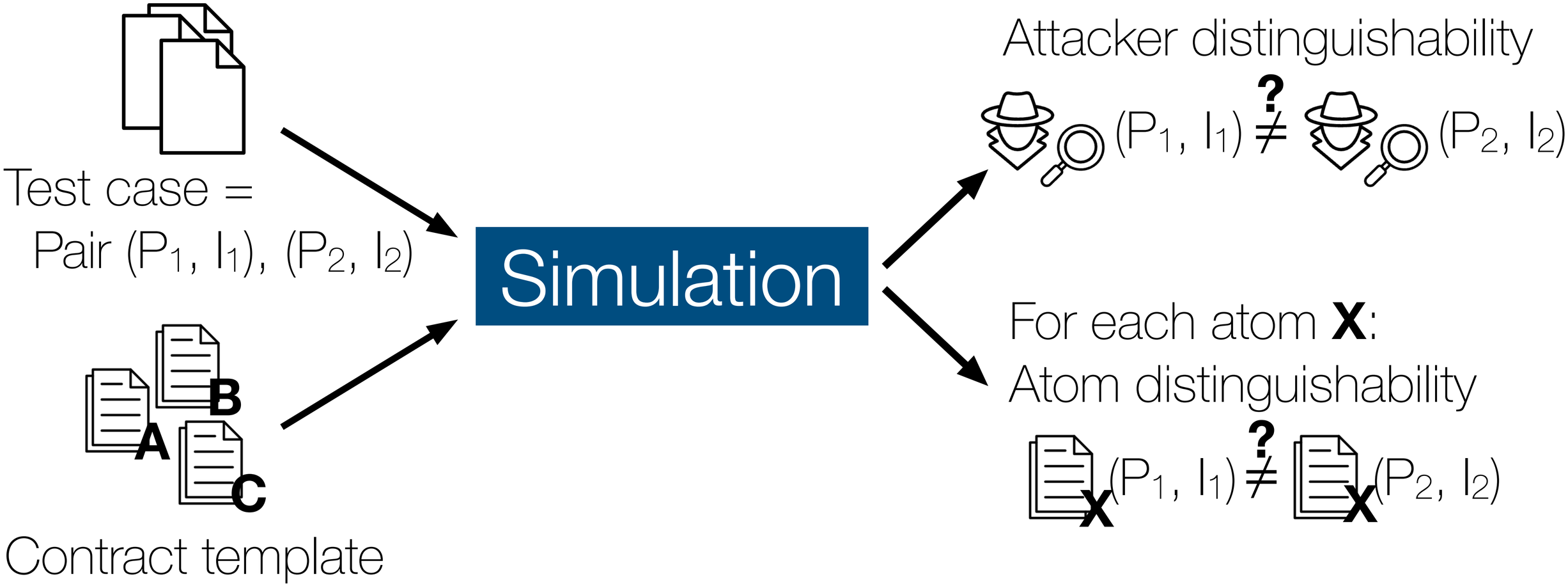
Contract Synthesis: Methodology (1/2)

Use test cases as proxy for contract satisfaction

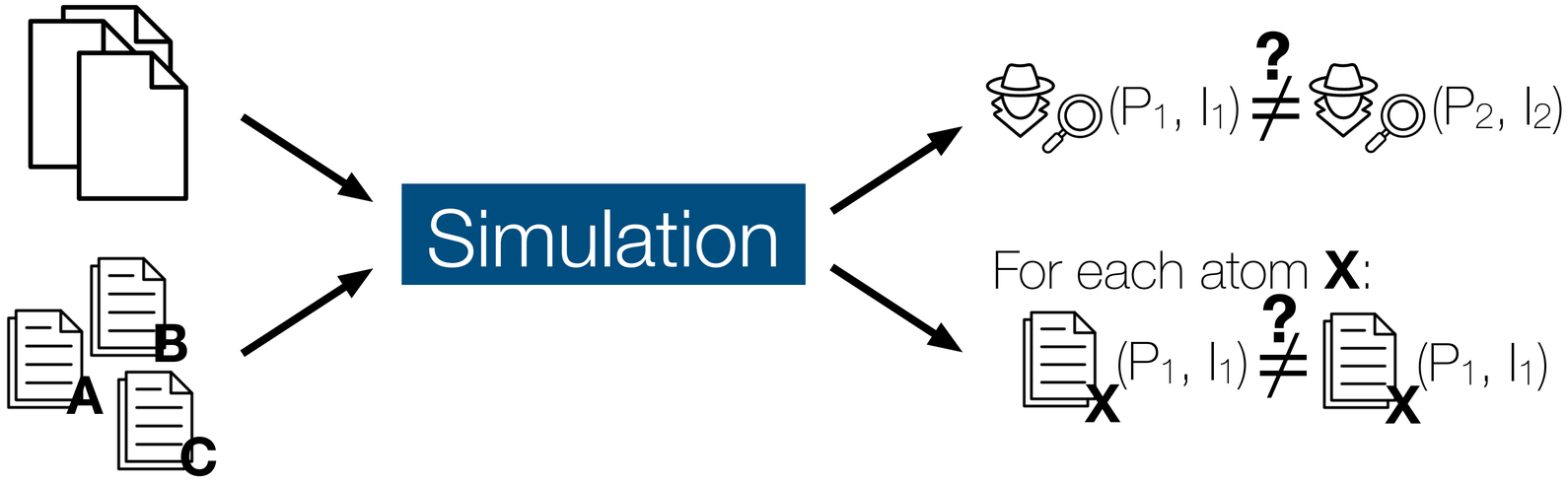


Contract Synthesis: Methodology (1/2)

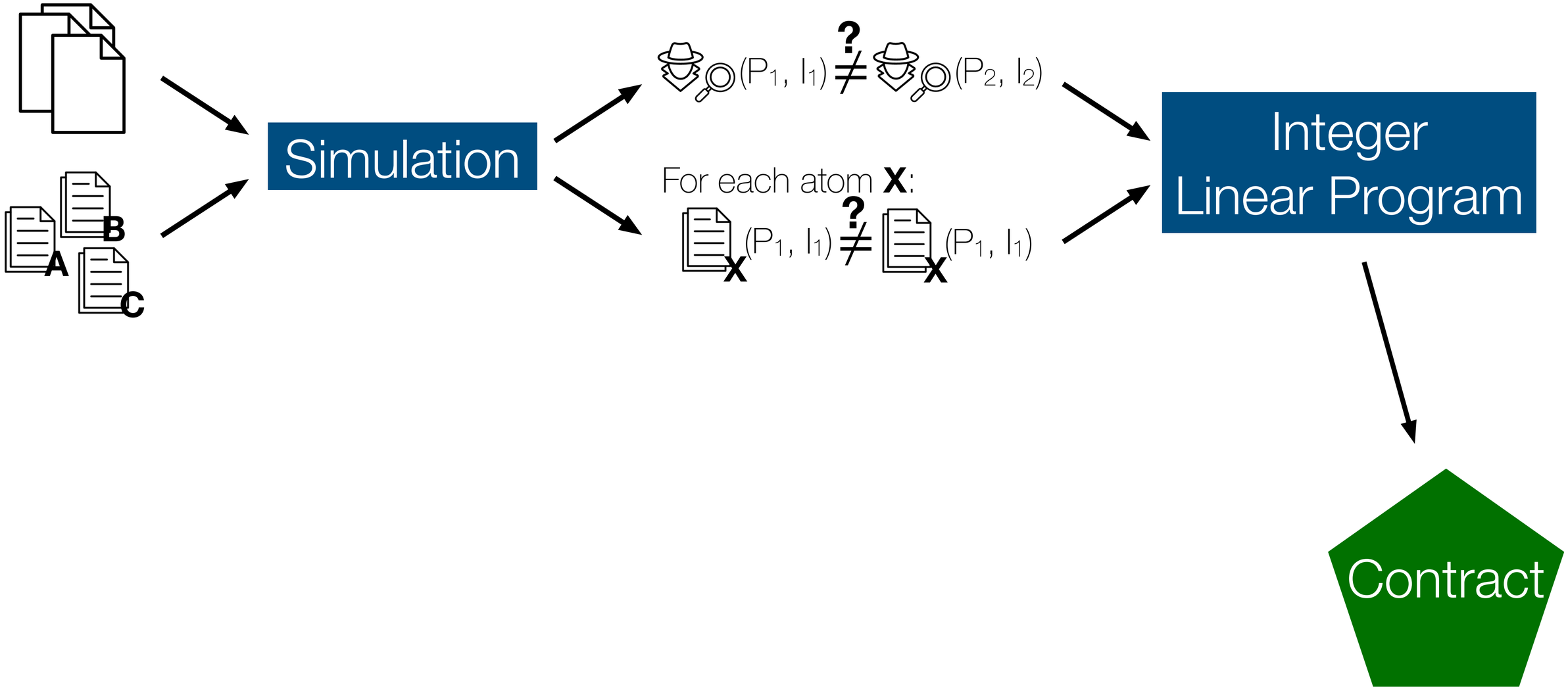
Use test cases as proxy for contract satisfaction



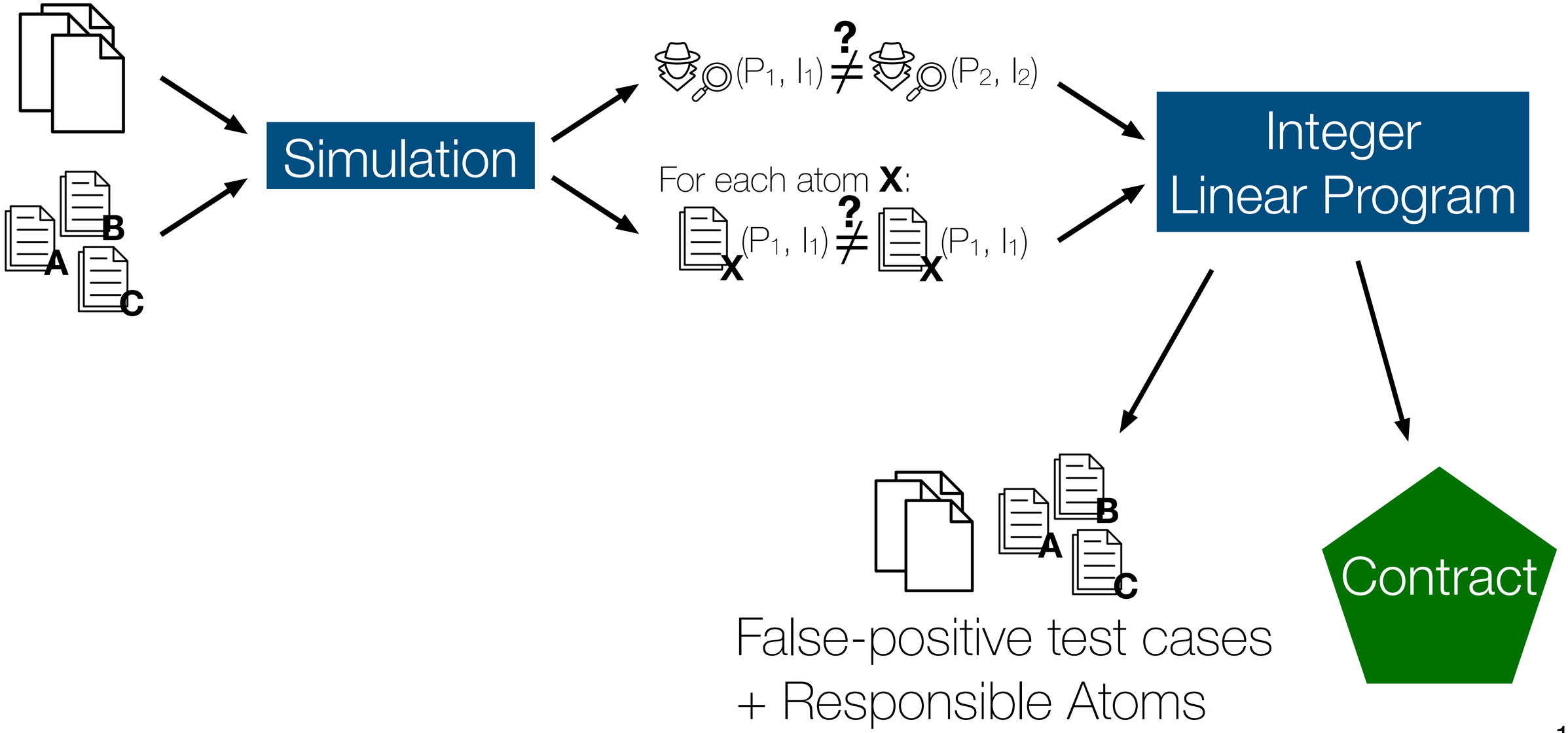
Contract Synthesis: Methodology (2/2)



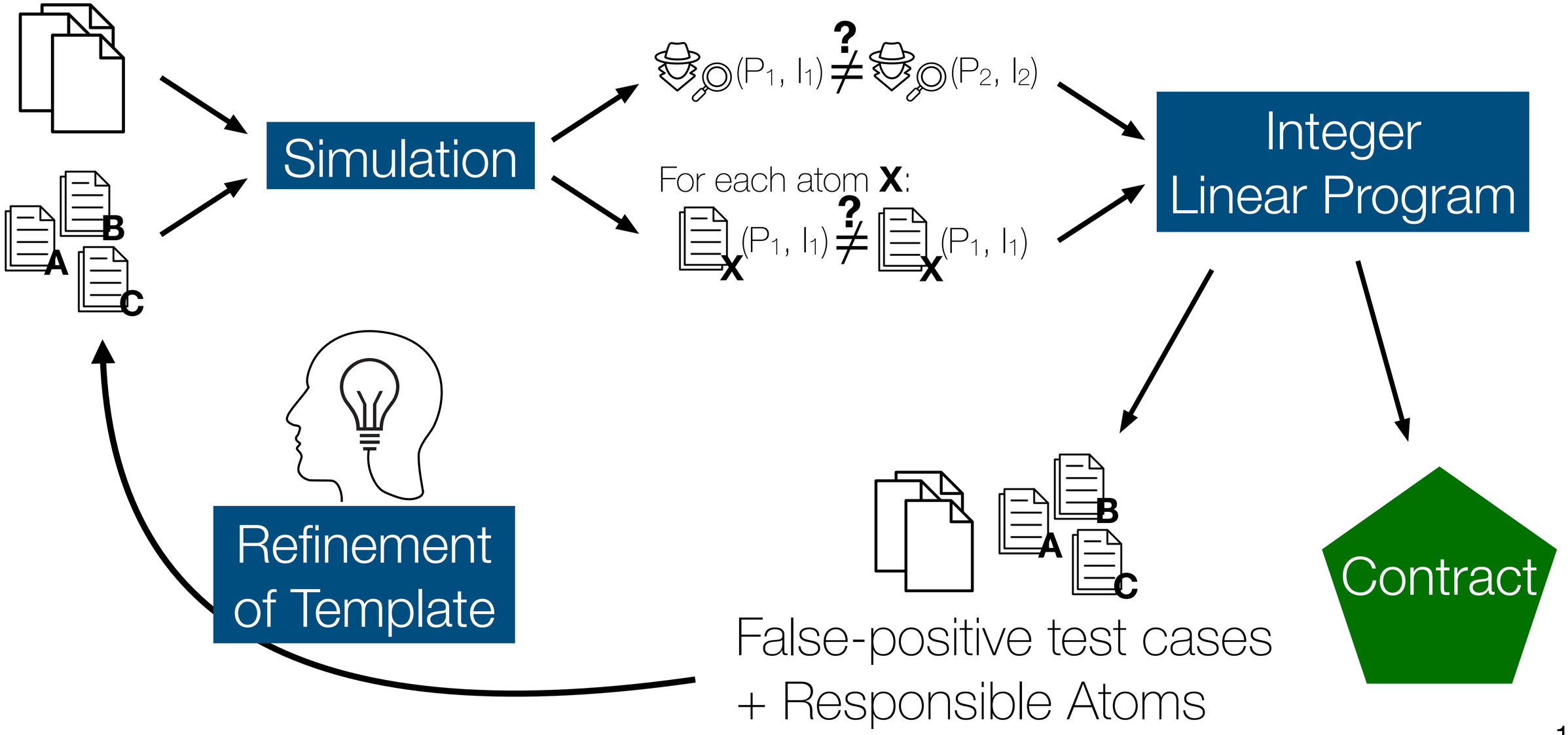
Contract Synthesis: Methodology (2/2)



Contract Synthesis: Methodology (2/2)



Contract Synthesis: Methodology (2/2)



Outline

1. Contracts and Contract Templates

2. Synthesis Goals and Methodology

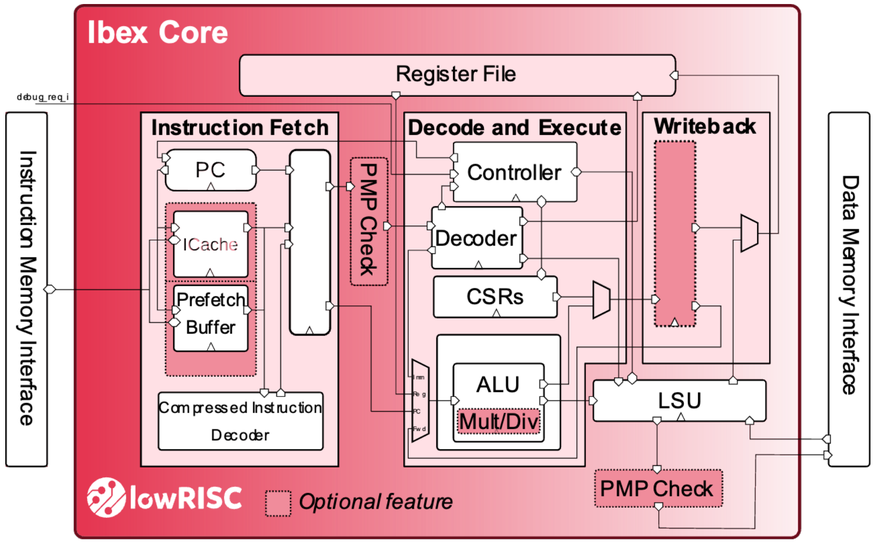
3. Some Experimental Results

Experimental Evaluation

Implemented using Icarus Verilog, Google OR-Tools, and RISC-V Formal Interface

Experimental Evaluation

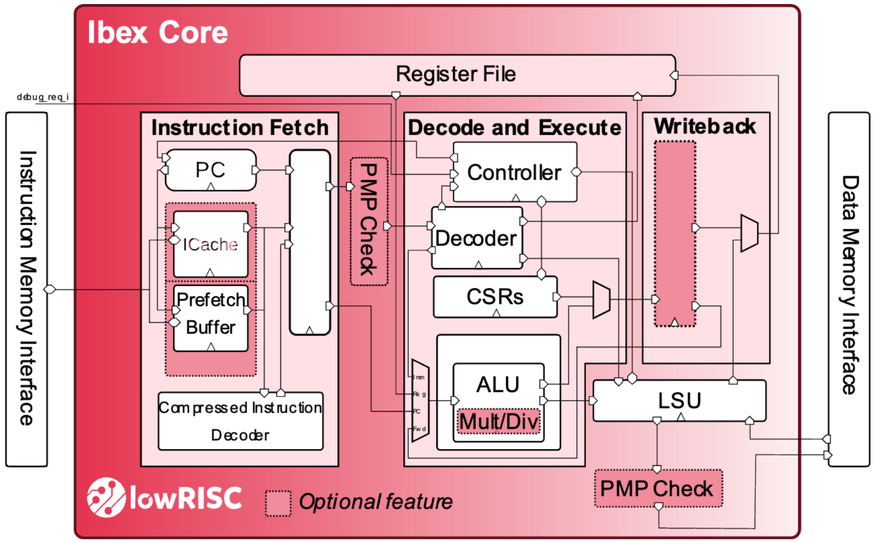
Implemented using Icarus Verilog, Google OR-Tools, and RISC-V Formal Interface



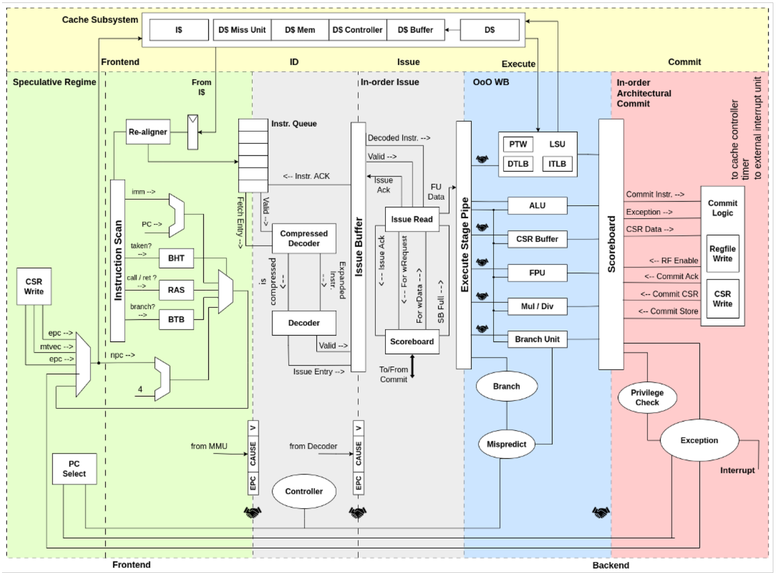
"**Ibex** is a production-quality open-source 32-bit RISC-V CPU core"

Experimental Evaluation

Implemented using Icarus Verilog, Google OR-Tools, and RISC-V Formal Interface



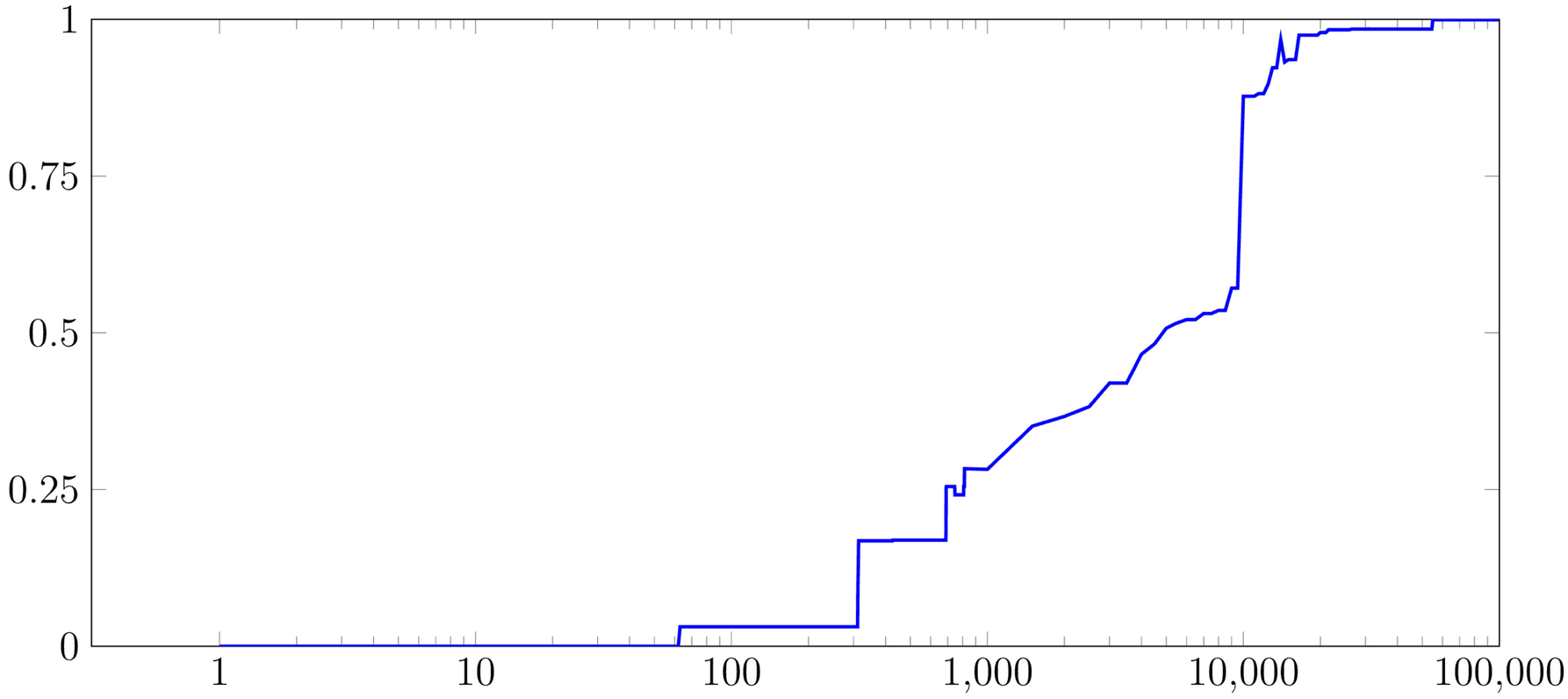
"**Ibex** is a production-quality open-source 32-bit RISC-V CPU core"



"**CVA6** is an application-class 6-stage RISC-V CPU capable of booting Linux"

Performance on Ibex - Sensitivity

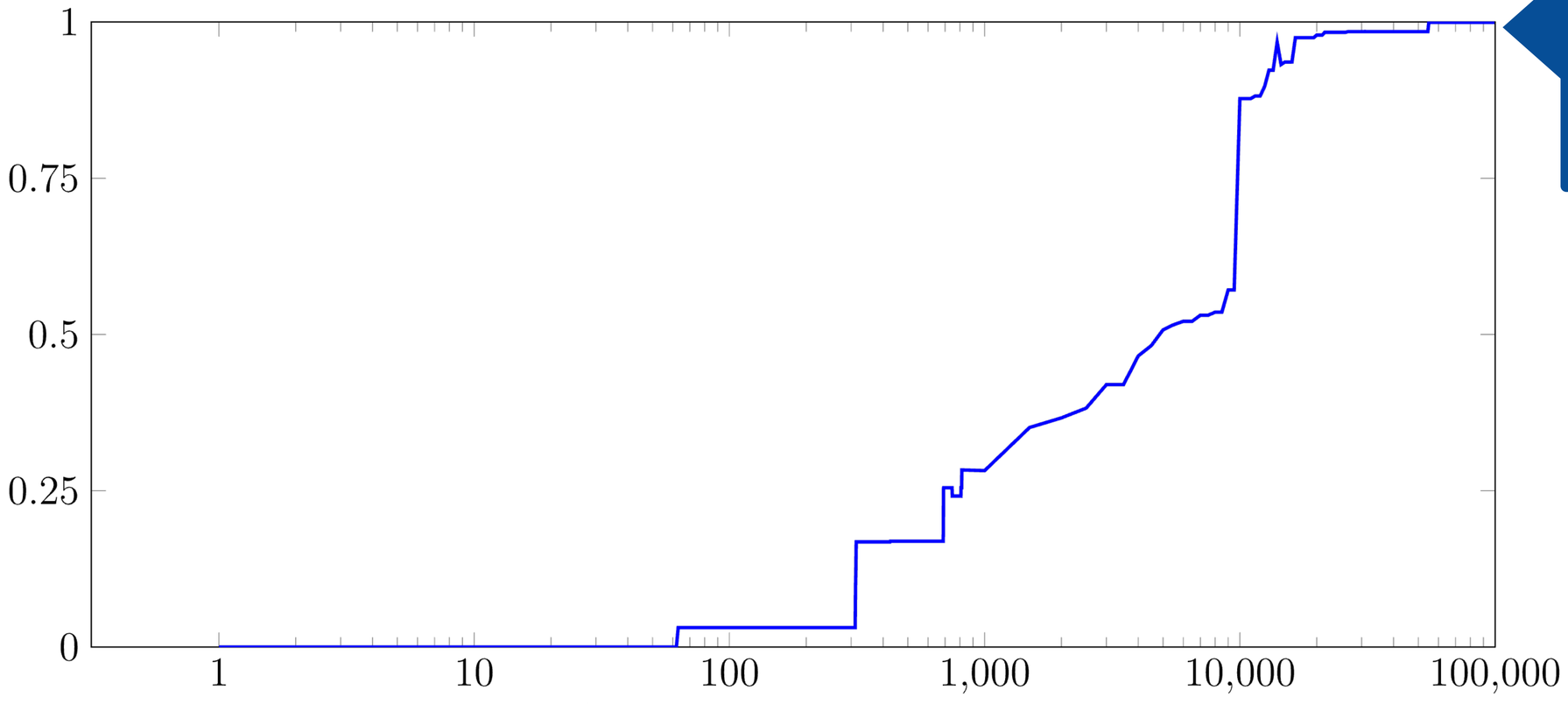
$$Sensitivity = \frac{\text{True Positives}}{\text{True Positives + False Negatives}}$$



Sensitivity w.r.t. a set of 2,000,000 test cases

Performance on Ibex - Sensitivity

$$Sensitivity = \frac{\text{True Positives}}{\text{True Positives + False Negatives}}$$

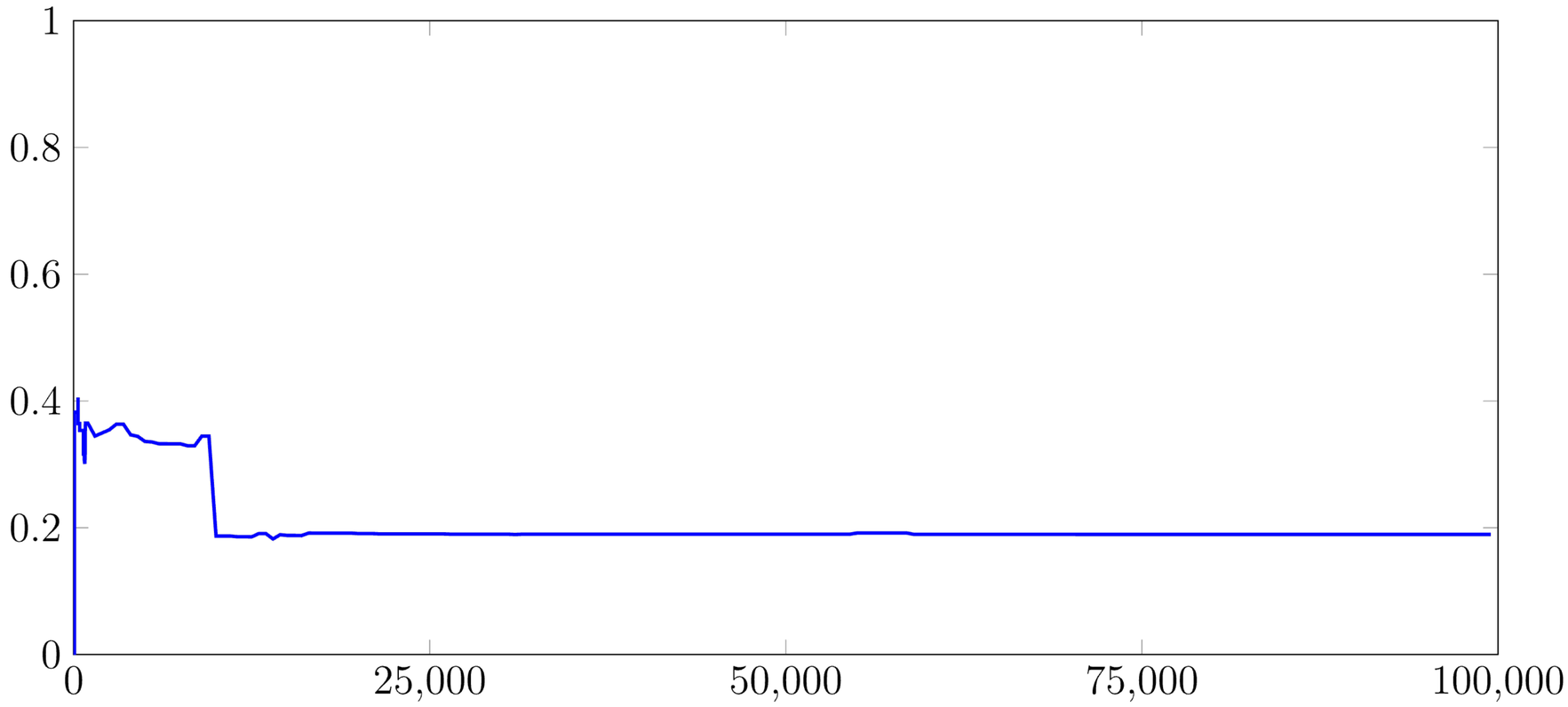


Final sensitivity: 99.94%

Sensitivity w.r.t. a set of 2,000,000 test cases

Performance on Ibex - Precision

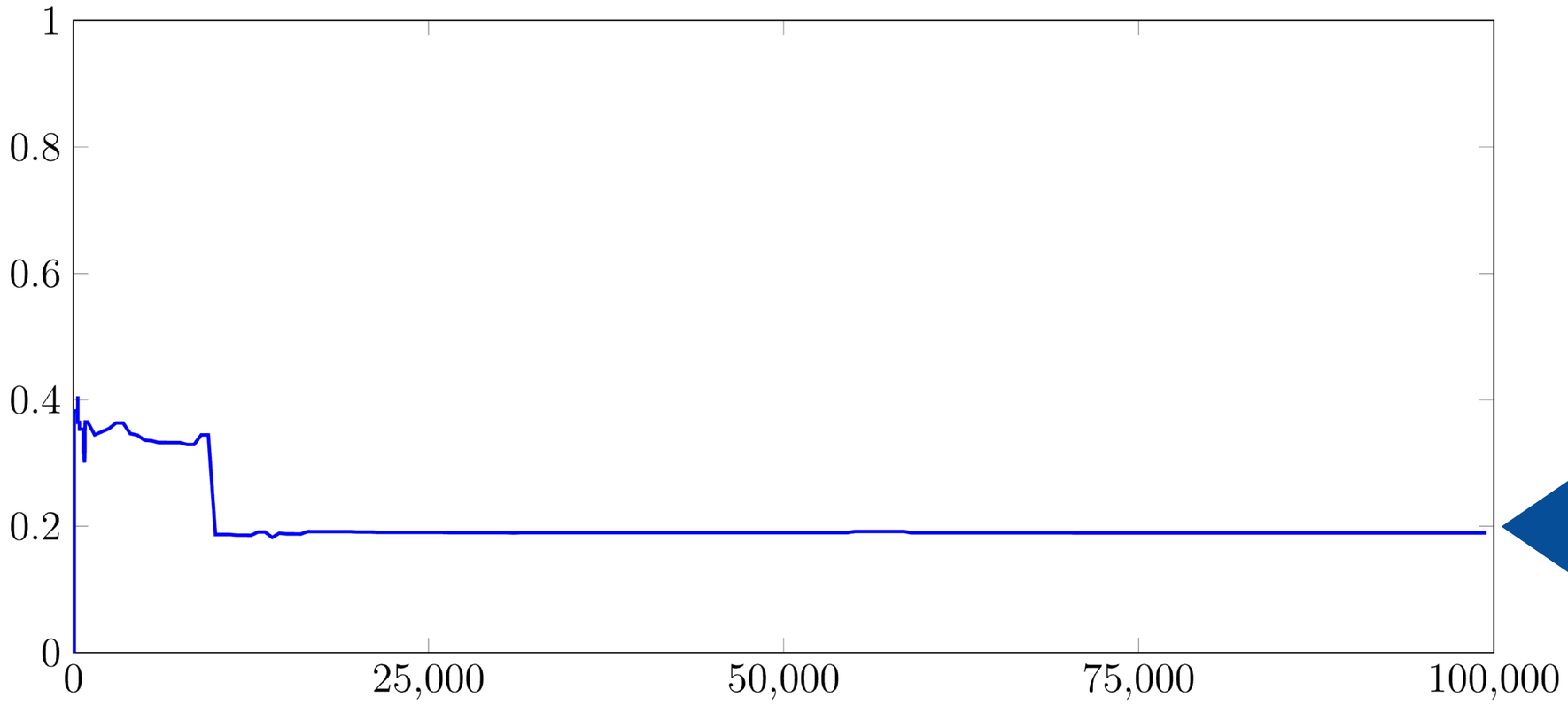
$$Precision = \frac{\text{True Positives}}{\text{True Positives + False Positives}}$$



Precision w.r.t. a set of 2,000,000 test cases

Performance on Ibex - Precision

$$Precision = \frac{\text{True Positives}}{\text{True Positives + False Positives}}$$



Final precision: 18.95%

Precision w.r.t. a set of 2,000,000 test cases

Refining Contract Template - Ibex Memory Interface

```
li a0, 0x100      a0 ← 0x100
lw a1, 0(a0)      a1 ← mem[a0]
```

Program 1

```
li a0, 0x102      a0 ← 0x102
lw a1, 0(a0)      a1 ← mem[a0]
```

Program 2

Refining Contract Template - Ibex Memory Interface

```
li a0, 0x100      a0 ← 0x100
lw a1, 0(a0)      a1 ← mem[a0]
```

Program 1

```
li a0, 0x102      a0 ← 0x102
lw a1, 0(a0)      a1 ← mem[a0]
```

Program 2

Slower!

Refining Contract Template - Ibex Memory Interface

```
li a0, 0x100      a0 ← 0x100
lw a1, 0(a0)      a1 ← mem[a0]
```

Program 1

```
li a0, 0x102      a0 ← 0x102
lw a1, 0(a0)      a1 ← mem[a0]
```

Program 2

Slower!

Address is not aligned

Refining Contract Template - Ibex Memory Interface

```
li a0, 0x100      a0 ← 0x100
lw a1, 0(a0)      a1 ← mem[a0]
```

Program 1

```
li a0, 0x102      a0 ← 0x102
lw a1, 0(a0)      a1 ← mem[a0]
```

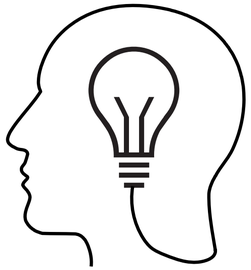
Program 2

Slower!

Address is not aligned

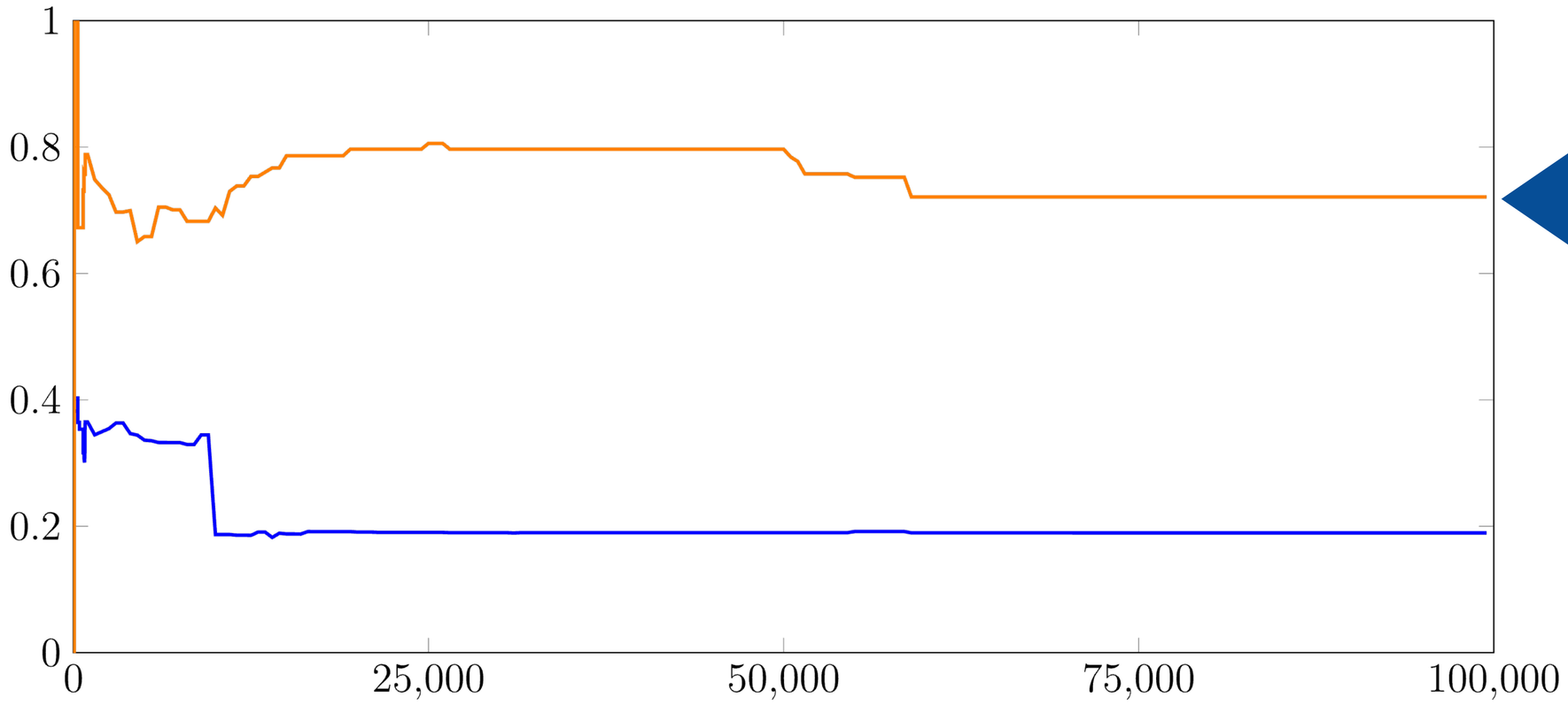
Refinement of template to capture:

- alignment of accesses
- branch outcomes
- data dependencies



Refined Contract Template - Precision

$$Precision = \frac{\text{True Positives}}{\text{True Positives + False Positives}}$$



Final precision: 72.11%

Precision w.r.t. a set of 2,000,000 test cases

Thank you for your attention!

Synthesizing Hardware-Software Leakage Contracts for RISC-V Open-Source Processors

Gideon Mohr
Saarland University
Saarland Informatics Campus
Saarbrücken, Germany

Marco Guarnieri
IMDEA Software Institute
Madrid, Spain

Jan Reineke
Saarland University
Saarland Informatics Campus
Saarbrücken, Germany

***Abstract*—Microarchitectural attacks compromise security by exploiting software-visible artifacts of microarchitectural optimizations such as caches and speculative execution. Defending against such attacks at the software level requires an appropriate abstraction at the instruction set architecture (ISA) level that captures microarchitectural leakage. Hardware-software leakage contracts have recently been proposed as such an abstraction.**

In this paper, we propose a semi-automatic methodology for synthesizing hardware-software leakage contracts for open-source microarchitectures. For a given ISA, our approach relies on human experts to (a) capture the space of possible contracts in the form of contract templates and (b) devise a test-case generation strategy to explore a microarchitecture's potential leakage. For a given implementation of an ISA, these two ingredients are then used to automatically synthesize the most precise leakage contract that is satisfied by the microarchitecture.

We have instantiated this methodology for the RISC-V ISA and applied it to the Ibex and CVA6 open-source processors. Our experiments demonstrate the practical applicability of the methodology and uncover subtle and unexpected leaks.

I. Introduction

Microarchitectural attacks [16], [26], [27], [36], [41] compromise the security of programs by exploiting software-visible artifacts of microarchitectural optimizations such as caches and speculative execution. Defending against such attacks in software is challenging: instruction set architectures (ISAs), the traditional hardware-software interface, abstract from microarchitectural details and thus do not give any guarantees w.r.t. these attacks.

Hardware-software leakage contracts [23], [39] (short: leakage contracts) have recently been proposed as a new security abstraction at the ISA level to fill this gap. Such contracts aim to capture possible microarchitectural side-channel leaks by associating leakage traces, i.e., sequences of leakage observations, with ISA-level executions. For example, a contract could expose the addresses of memory instructions as leakage observations to capture cache leaks. Similarly, a contract could expose ... instructions to capture leakage via ... that faithfully

However, most of today's processor designs lack precise specifications of microarchitectural leakage. In this work, we close this gap by proposing a semi-automatic methodology for synthesizing leakage contracts from open-source processor designs.

Our methodology consists of four steps:

1) **Definition of Contract Template**. A human expert determines a set of *contract atoms* that capture potential instruction-level leakage observations. For example, a contract atom may expose the value of the register operand of a memory instruction. The set of all contract atoms forms the *contract template*, and any of its subsets is a candidate *contract*.

2) **Test-Case Generation**. A human expert devises a test-case generation strategy. Each *test case* consists of two ISA-level programs with fixed data inputs. These test cases are used to exercise and analyze a processor's microarchitectural leakage.

3) **Evaluation of Test Cases**. Test cases are automatically evaluated on the target processor design to determine which test cases are *distinguishable*, *i.e.*, lead to distinguishable executions for a given microarchitectural attacker. Distinguishable test cases expose actual leaks that need to be accounted for at contract level. Test cases are also automatically evaluated on the contract template to derive the set of *distinguishing atoms*, i.e., those atoms that distinguish the two programs. For instance, an atom exposing the value of register operands for memory instructions will distinguish programs accessing different addresses.

4) **Automatic Contract Synthesis**. Based on the results of the test-case evaluation, a contract, *i.e.* a set of contract atoms, is automatically synthesized from the distinguishing atoms associated with attacker-distinguishable test case. Our approach ensures that the synthesized contract is *satisfied* by the processor design on all test cases, *i.e.*, it captures all leaks exposed by the test cases on the processor. Moreover, it also ensures that the synthesized contract is the *most precise* such contract, *i.e.*, it distinguishes the fewest attacker-indistinguishable test cases.

The proliferation of the open-source RISC-V ISA [13] and its growing ecosystem is promising in this context for two reasons: (1) The simplicity and modularity of the ISA provides ... lation for the definition of contract atoms. (2) The ... RISC-V processor designs ... targets for contract ... we

Read the paper (to appear at DATE 2024):
https://arxiv.org/abs/2401.09383

Play with the artifact:
https://github.com/hw-sw-contracts/riscv-contract-synthesis
https://zenodo.org/records/10491534