UNIVERSITÄT
DES
SAARLANDES

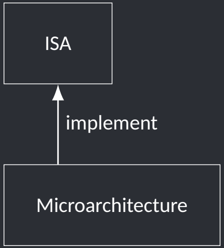
Automatic Inference of Hardware-Software Contracts for Open-Source Processors

Gideon Mohr

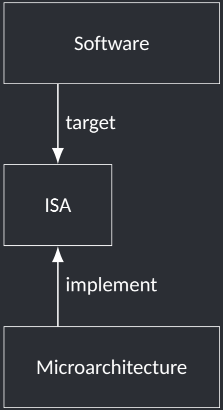
Saarland University

Hardware-Software Contracts

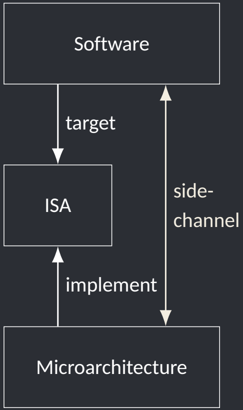
Background



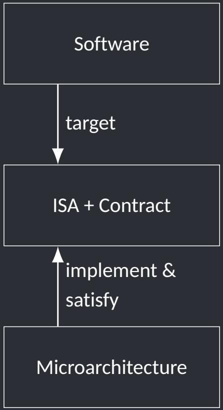
Background



Background

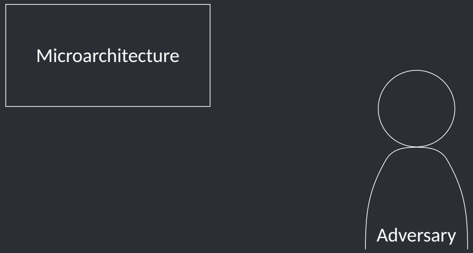

Software
target
ISA
side-
channel
implement
Microarchitecture

Background


Software
target
ISA + Contract
implement &
satisfy
Microarchitecture

What are HW/SW Contracts?

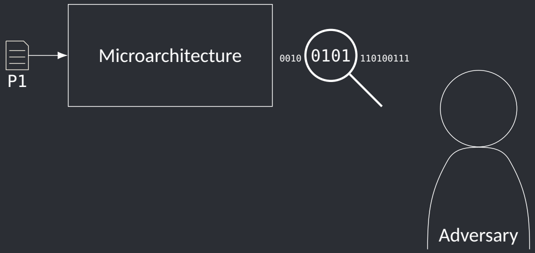
What are HW/SW Contracts?



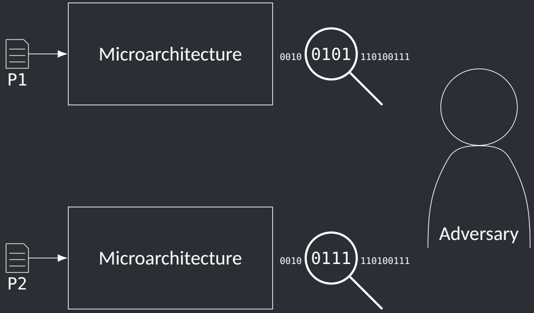
Microarchitecture

Adversary

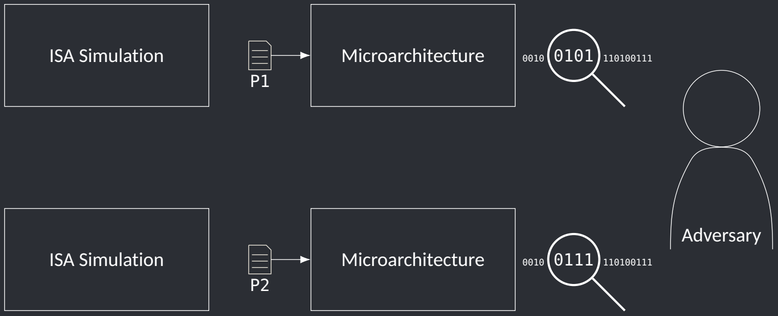
What are HW/SW Contracts?



What are HW/SW Contracts?



What are HW/SW Contracts?



ISA Simulation

P1

Microarchitecture

0010 0101 110100111

ISA Simulation

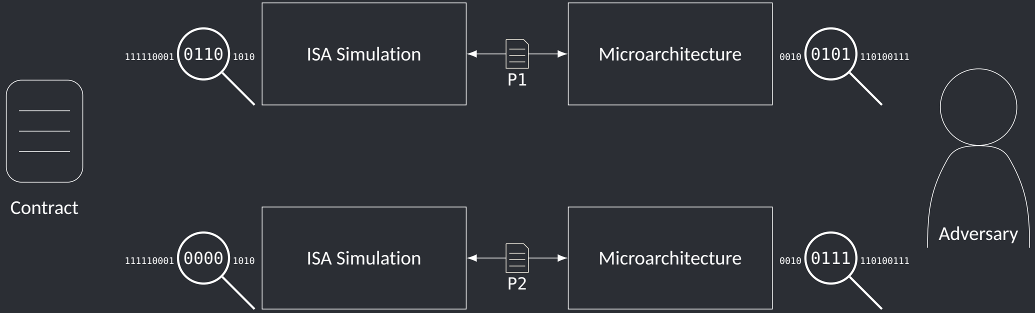
P2

Microarchitecture

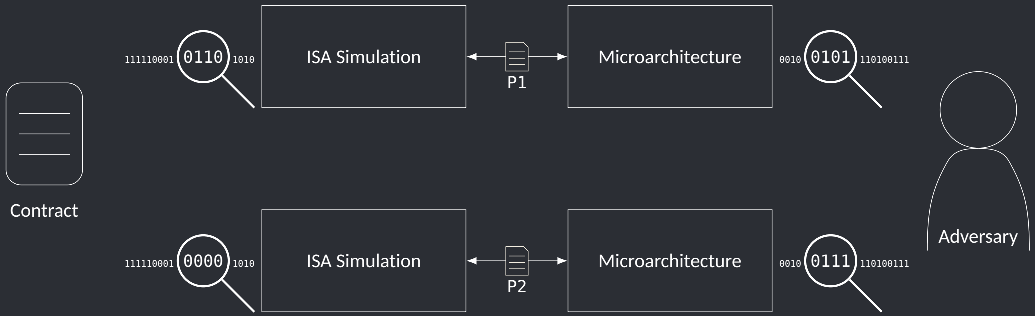
0010 0111 110100111

Adversary

What are HW/SW Contracts?

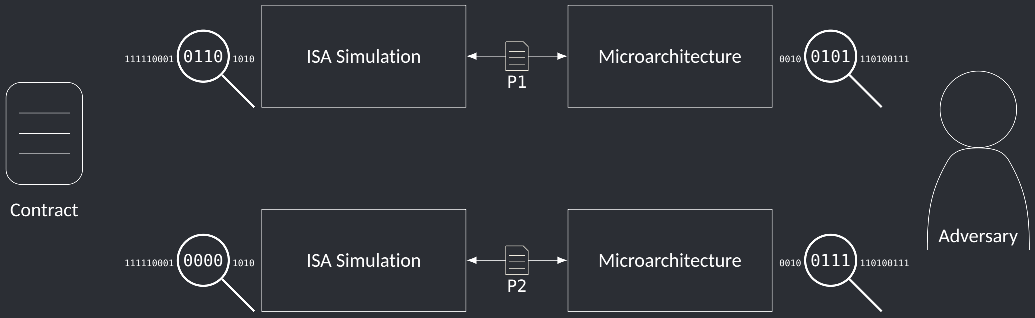


What are HW/SW Contracts?



$$\mathit{ADV}(\mathrm{P1}) \neq \mathit{ADV}(\mathrm{P2}) \Rightarrow \mathit{CTR}(\mathrm{P1}) \neq \mathit{CTR}(\mathrm{P2})$$

What are HW/SW Contracts?



$$ADV(\mathrm{P1}) \neq ADV(\mathrm{P2}) \Rightarrow CTR(\mathrm{P1}) \neq CTR(\mathrm{P2})$$

$$\mathrm{MARCH} \models_{\mathrm{ADV}} \mathrm{CTR}$$

Contract Generation

Definition (Hardware-Software Contract Generation)

Given an instruction set architecture ISA, a microarchitecture MARCH that implements ISA, and an adversary model ADV suitable for MARCH, find a hardware-software contract CTR that satisfies the following:

Definition (Hardware-Software Contract Generation)

Given an instruction set architecture ISA, a microarchitecture MARCH that implements ISA, and an adversary model ADV suitable for MARCH, find a hardware-software contract CTR that satisfies the following:

1. Contract satisfaction:

$$\mathtt{MARCH} \vDash_{\mathtt{ADV}} \mathtt{CTR}$$

Definition (Hardware-Software Contract Generation)

Given an instruction set architecture $\mathtt{ISA}$, a microarchitecture $\mathtt{MARCH}$ that implements $\mathtt{ISA}$, and an adversary model $\mathtt{ADV}$ suitable for $\mathtt{MARCH}$, find a hardware-software contract $\mathtt{CTR}$ that satisfies the following:

1. Contract satisfaction:

$$\mathtt{MARCH} \vDash_{\mathtt{ADV}} \mathtt{CTR}$$

2. Least contract:

$$\forall \mathtt{CTR}'.\ \mathtt{MARCH} \vDash_{\mathtt{ADV}} \mathtt{CTR}'$$
$$\Rightarrow \mathtt{CTR} \leq \mathtt{CTR}'$$

Definition (Hardware-Software Contract Generation)

Given an instruction set architecture $\mathtt{ISA}$, a microarchitecture $\mathtt{MARCH}$ that implements $\mathtt{ISA}$, an adversary model $\mathtt{ADV}$ suitable for $\mathtt{MARCH}$, and a function $p : C \rightarrow \mathbb{N}$, find a hardware-software contract $\mathtt{CTR}$ that satisfies the following:

1. Contract satisfaction:
$$\mathtt{MARCH} \vDash_{\mathtt{ADV}} \mathtt{CTR}$$
2. Most precise contract:
$$\begin{aligned} &\forall \mathtt{CTR}'.\ \mathtt{MARCH} \vDash_{\mathtt{ADV}} \mathtt{CTR}' \\ &\Rightarrow p(\mathtt{CTR}) \leq p(\mathtt{CTR}') \end{aligned}$$

Determining p

Determining *p*

- Select least contract if a least contract exists

Determining p

- Select least contract if a least contract exists
- Select a minimal contract

Determining *p*

- Select least contract if a least contract exists
- Select a minimal contract
- Additional parameter: Precision of the contract

Determining *p*

- Select least contract if a least contract exists
- Select a minimal contract
- Additional parameter: Precision of the contract

$$\text{Precision} = \frac{\text{True Positive}}{\text{Predicted Positive}} \mathrel{\widehat{=}} \frac{\text{Adversary Distinguishable}}{\text{Contract Distinguishable}}$$

Definition (Hardware-Software Contract Generation)

Given an instruction set architecture $\mathtt{ISA}$, a microarchitecture $\mathtt{MARCH}$ that implements $\mathtt{ISA}$, an adversary model $\mathtt{ADV}$ suitable for $\mathtt{MARCH}$, and a function $p : C \longrightarrow \mathbb{N}$ find a hardware-software contract $\mathtt{CTR}$ that satisfies the following:

1. Contract satisfaction:

$$\mathtt{MARCH} \vDash_{\mathtt{ADV}} \mathtt{CTR}$$

2. Most precise contract:

$$\begin{aligned} &\forall \mathtt{CTR}'.\ \mathtt{MARCH} \vDash_{\mathtt{ADV}} \mathtt{CTR}' \\ &\Rightarrow p(\mathtt{CTR}) \leq p(\mathtt{CTR}') \end{aligned}$$

Definition (Hardware-Software Contract Generation)

Given an instruction set architecture $\texttt{ISA}$, a microarchitecture $\texttt{MARCH}$ that implements $\texttt{ISA}$, an adversary model $\texttt{ADV}$ suitable for $\texttt{MARCH}$, a space of possible contracts C, and a function $p : C \longrightarrow \mathbb{N}$ find a hardware-software contract $\texttt{CTR} \in C$ that satisfies the following:

1. Contract satisfaction:

$$\texttt{MARCH} \vDash_{\texttt{ADV}} \texttt{CTR}$$

2. Most precise contract:

$$\begin{aligned} &\forall \texttt{CTR}' \in C.\ \texttt{MARCH} \vDash_{\texttt{ADV}} \texttt{CTR}' \\ &\qquad \Rightarrow p(\texttt{CTR}) \leq p(\texttt{CTR}') \end{aligned}$$

Contract Template C

Contract Template C

- Instruction-Level Contract

Contract Template C

- Instruction-Level Contract

  ⟶ Leakage associated to the executed instruction

Contract Template C

- Instruction-Level Contract

  ⟶ Leakage associated to the executed instruction
- Various elements of the architectural state could leak: immediate, registers, memory

Contract Template C

- Instruction-Level Contract
  → Leakage associated to the executed instruction
- Various elements of the architectural state could leak: immediate, registers, memory
- Observation: Instructions of the same type usually behave similarly

Contract Template C

- Instruction-Level Contract
  → Leakage associated to the executed instruction
- Various elements of the architectural state could leak: immediate, registers, memory
- Observation: Instructions of the same type usually behave similarly
- Idea: Leak certain architectural values depending on the type of the executed instruction

Contract Template C

- Instruction-Level Contract
  → Leakage associated to the executed instruction
- Various elements of the architectural state could leak: immediate, registers, memory
- Observation: Instructions of the same type usually behave similarly
- Idea: Leak certain architectural values depending on the type of the executed instruction
- Example: `LW: IMM, REG_RS1`

Definition (Hardware-Software Contract Generation)

Given an instruction set architecture $\mathtt{ISA}$, a microarchitecture $\mathtt{MARCH}$ that implements $\mathtt{ISA}$, an adversary model $\mathtt{ADV}$ suitable for $\mathtt{MARCH}$, a space of possible contracts C, and a function $p : C \rightarrow \mathbb{N}$, find a hardware-software contract $\mathtt{CTR} \in C$ that satisfies the following:

1. Contract satisfaction:

$$\mathtt{MARCH} \vDash_{\mathtt{ADV}} \mathtt{CTR}$$

2. Most precise contract:

$$\begin{aligned} &\forall \mathtt{CTR}' \in C.\ \mathtt{MARCH} \vDash_{\mathtt{ADV}} \mathtt{CTR}' \\ &\qquad \Rightarrow p(\mathtt{CTR}) \leq p(\mathtt{CTR}') \end{aligned}$$

Definition (Hardware-Software Contract Candidate Generation)

Given an instruction set architecture $\mathtt{ISA}$, a microarchitecture $\mathtt{MARCH}$ that implements $\mathtt{ISA}$, an adversary model $\mathtt{ADV}$ suitable for $\mathtt{MARCH}$, a space of possible contracts C, a function $p : C \longrightarrow \mathbb{N}$, and a set of test cases $\mathtt{TC}$, find a hardware-software contract candidate $\mathtt{CTR} \in C$ that satisfies the following:

1. Contract candidate satisfaction:

$$\mathtt{MARCH} \models^{\mathtt{TC}}_{\mathtt{ADV}} \mathtt{CTR}$$

2. Most precise contract:

$$\forall \mathtt{CTR}' \in C.\ \mathtt{MARCH} \models^{\mathtt{TC}}_{\mathtt{ADV}} \mathtt{CTR}' \\ \Rightarrow p(\mathtt{CTR}) \leq p(\mathtt{CTR}')$$

Algorithm

```
Distinguishable, Indistinguishable ← EmptyList( )
for all TC in TC[ ] do
    TRACE, ADVDistinguishable ← simulate(MARCH, ADV, TC)
    OBS ← analyze(TRACE)
    if ADVDistinguishable then
        Distinguishable.append(OBS)
    else
        Indistinguishable.append(OBS)
    end if
end for
CTR ← compute(Distinguishable, Indistinguishable)
```

Algorithm

```
Distinguishable, Indistinguishable ← EmptyList( )
for all TC in TC[ ] do
    TRACE, ADVDistinguishable ← simulate(MARCH, ADV, TC)
    OBS ← analyze(TRACE)
    if ADVDistinguishable then
        Distinguishable.append(OBS)
    else
        Indistinguishable.append(OBS)
    end if
end for
CTR ← compute(Distinguishable, Indistinguishable)
```

Algorithm

```
Distinguishable, Indistinguishable ← EmptyList( )
for all TC in TC[ ] do
    TRACE, ADVDistinguishable ← simulate(MARCH, ADV, TC)
    OBS ← analyze(TRACE)
    if ADVDistinguishable then
        Distinguishable.append(OBS)
    else
        Indistinguishable.append(OBS)
    end if
end for
CTR ← compute(Distinguishable, Indistinguishable)
```

Algorithm

```
Distinguishable, Indistinguishable ← EmptyList( )
for all TC in TC[ ] do
    TRACE, ADVDistinguishable ← simulate(MARCH, ADV, TC)
    OBS ← analyze(TRACE)
    if ADVDistinguishable then
        Distinguishable.append(OBS)
    else
        Indistinguishable.append(OBS)
    end if
end for
CTR ← compute(Distinguishable, Indistinguishable)
```

Extraction of Possible Observations

Extraction of Possible Observations

- Extract the sequence of architectural states from the trace

Extraction of Possible Observations

- Extract the sequence of architectural states from the trace
- Compare values according to the contract template

Extraction of Possible Observations

- Extract the sequence of architectural states from the trace
- Compare values according to the contract template

Core 1	`type`	
	`RS1`	
	`RS2`	
	`REG_RS1`	
Core 2	`type`	
	`RS1`	
	`RS2`	
	`REG_RS1`	

Extraction of Possible Observations

- Extract the sequence of architectural states from the trace
- Compare values according to the contract template

Core 1	type	add
	RS1	r4
	RS2	r2
	REG_RS1	0xAB
Core 2	type	add
	RS1	r4
	RS2	r2
	REG_RS1	0xCD

Extraction of Possible Observations

- Extract the sequence of architectural states from the trace
- Compare values according to the contract template

Core 1	type	add
	RS1	r4
	RS2	r2
	REG_RS1	0xAB
Core 2	type	add
	RS1	r4
	RS2	r2
	REG_RS1	0xCD

Possible Observations:

- `ADD: REG_RS1`

Extraction of Possible Observations

- Extract the sequence of architectural states from the trace
- Compare values according to the contract template

Core 1	type	add	or
	RS1	r4	r8
	RS2	r2	r9
	REG_RS1	0xAB	0x12
Core 2	type	add	or
	RS1	r4	r10
	RS2	r2	r9
	REG_RS1	0xCD	0x12

Possible Observations:

- `ADD: REG_RS1`

Extraction of Possible Observations

- Extract the sequence of architectural states from the trace
- Compare values according to the contract template

Core 1	type	add	or
	RS1	r4	r8
	RS2	r2	r9
	REG_RS1	0xAB	0x12
Core 2	type	add	or
	RS1	r4	r10
	RS2	r2	r9
	REG_RS1	0xCD	0x12

Possible Observations:

- ADD: REG_RS1
- OR: RS1

Extraction of Possible Observations

- Extract the sequence of architectural states from the trace
- Compare values according to the contract template

Core 1	type	add	or	. . .
	RS1	r4	r8	
	RS2	r2	r9	
	REG_RS1	0xAB	0x12	
Core 2	type	add	or	. . .
	RS1	r4	r10	
	RS2	r2	r9	
	REG_RS1	0xCD	0x12	

Possible Observations:

- `ADD: REG_RS1`
- `OR: RS1`
- . . .

Algorithm

```
Distinguishable, Indistinguishable ← EmptyList( )
for all TC in TC[ ] do
    TRACE, ADVDistinguishable ← simulate(MARCH, ADV, TC)
    OBS ← analyze(TRACE)
    if ADVDistinguishable then
        Distinguishable.append(OBS)
    else
        Indistinguishable.append(OBS)
    end if
end for
CTR ← compute(Distinguishable, Indistinguishable)
```

Algorithm

```
Distinguishable, Indistinguishable ← EmptyList( )
for all TC in TC[ ] do
    TRACE, ADVDistinguishable ← simulate(MARCH, ADV, TC)
    OBS ← analyze(TRACE)
    if ADVDistinguishable then
        Distinguishable.append(OBS)
    else
        Indistinguishable.append(OBS)
    end if
end for
CTR ← compute(Distinguishable, Indistinguishable)
```

Algorithm

```
Distinguishable, Indistinguishable ← EmptyList( )
for all TC in TC[ ] do
    TRACE, ADVDistinguishable ← simulate(MARCH, ADV, TC)
    OBS ← analyze(TRACE)
    if ADVDistinguishable then
        Distinguishable.append(OBS)
    else
        Indistinguishable.append(OBS)
    end if
end for
CTR ← compute(Distinguishable, Indistinguishable)
```

Evaluation

Examined Cores

Examined Cores

Ibex

CVA6

Examined Cores

Ibex

- `RV32IMC ISA`, `I` and `M` evaluated
- 3 Stages
- Optional caches (disabled for evaluation)

CVA6

Examined Cores

Ibex

- `RV32IMC ISA`, `I` and `M` evaluated
- 3 Stages
- Optional caches (disabled for evaluation)

CVA6

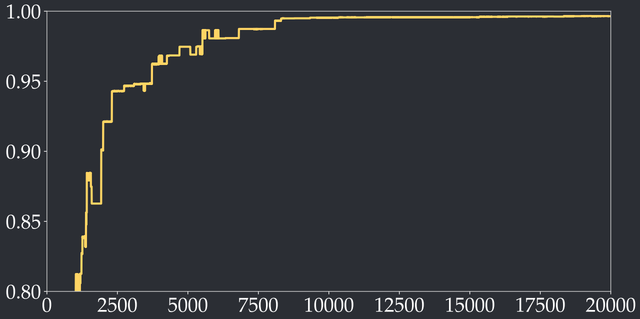
- `RV32IMA ISA`, `I` and `M` evaluated
- 6 stages
- In-order CPU
- Instruction Cache
- Branch prediction
- Multiple execution units
- Ready for FPGA deployment

Sets of Test Cases

	Size	Adversary Distinguishable			
		Ibex		CVA6	
Training Set	20,000	1421	7.1%	1055	5.2%
Evaluation Set	100,000	7035	7.0%	5573	5.5%

Sensitivity

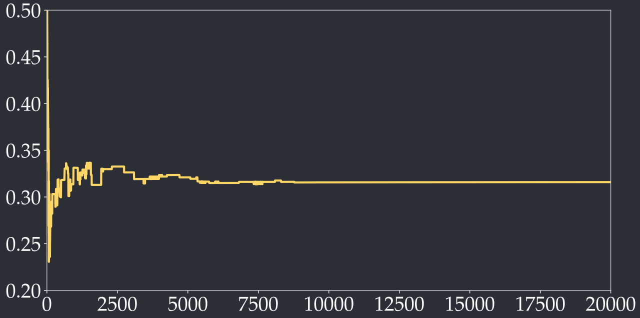
	Size	Adversary Distinguishable			
		Ibex		CVA6	
Training Set	20,000	1421	7.1%	1055	5.2%
Evaluation Set	100,000	7035	7.0%	5573	5.5%



$$\text{Sensitivity} = \frac{\text{True Positive}}{\text{Actual Positive}}$$

Precision

	Size	Adversary Distinguishable			
		Ibex		CVA6	
Training Set	20,000	1421	7.1%	1055	5.2%
Evaluation Set	100,000	7035	7.0%	5573	5.5%



$$\text{Precision} = \frac{\text{True Positive}}{\text{Predicted Positive}}$$

The Generated Contract

	Size	Adversary Distinguishable			
		Ibex		CVA6	
Training Set	20,000	1421	7.1%	1055	5.2%
Evaluation Set	100,000	7035	7.0%	5573	5.5%

Example: Loads on the Ibex core

```
LW: IMM
LW: REG_RS1
```

The Generated Contract

	Size	Adversary Distinguishable			
		Ibex		CVA6	
Training Set	20,000	1421	7.1%	1055	5.2%
Evaluation Set	100,000	7035	7.0%	5573	5.5%

Example: Loads on the Ibex core

```
LW: IMM
LW: REG_RS1
```

Example: Branches on the CVA6 core:

```
BGE: IMM
BGE: REG_RS1
BGE: REG_RS2
```

Computation Time

On an Intel Core i7-8700 CPU @ 3.20GHz with 12 threads and 16 GB of RAM:

	Ibex	CVA6
Compilation Time		
Simulation Time[1]		
Extraction of Possible Observations[1]		
Contract Candidate Computation[2]		
Total Contract Candidate Generation Time[2]		

[1] on average, per test case. [2] using the training set with 20,000 test cases, multi-threaded.

Computation Time

On an Intel Core i7-8700 CPU @ 3.20GHz with 12 threads and 16 GB of RAM:

	Ibex	CVA6
Compilation Time	3.4s	
Simulation Time[1]	83ms	
Extraction of Possible Observations[1]	3ms	
Contract Candidate Computation[2]	3.2s	
Total Contract Candidate Generation Time[2]		

[1] on average, per test case. [2] using the training set with 20,000 test cases, multi-threaded.

Computation Time

On an Intel Core i7-8700 CPU @ 3.20GHz with 12 threads and 16 GB of RAM:

	Ibex	CVA6
Compilation Time	3.4s	
Simulation Time[1]	83ms	
Extraction of Possible Observations[1]	3ms	
Contract Candidate Computation[2]	3.2s	
Total Contract Candidate Generation Time[2]	7.5min	

[1] on average, per test case. [2] using the training set with 20,000 test cases, multi-threaded.

Computation Time

On an Intel Core i7-8700 CPU @ 3.20GHz with 12 threads and 16 GB of RAM:

	Ibex	CVA6
Compilation Time	3.4s	20.0s
Simulation Time[1]	83ms	2.8s
Extraction of Possible Observations[1]	3ms	21ms
Contract Candidate Computation[2]	3.2s	1.3s
Total Contract Candidate Generation Time[2]	7.5min	

[1] on average, per test case. [2] using the training set with 20,000 test cases, multi-threaded.

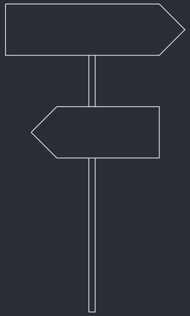
Computation Time

On an Intel Core i7-8700 CPU @ 3.20GHz with 12 threads and 16 GB of RAM:

	Ibex	**CVA6**
Compilation Time	3.4s	20.0s
Simulation Time[1]	83ms	2.8s
Extraction of Possible Observations[1]	3ms	21ms
Contract Candidate Computation[2]	3.2s	1.3s
Total Contract Candidate Generation Time[2]	7.5min	3.25h

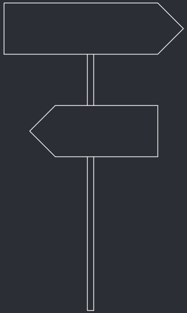
[1] on average, per test case. [2] using the training set with 20,000 test cases, multi-threaded.

Conclusion



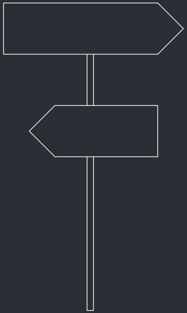
Conclusion

- Contract generation is generally possible



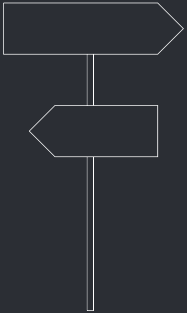
Conclusion

- Contract generation is generally possible
- Few test cases result in a relatively accurate contract, however, the contract slowly keeps getting better

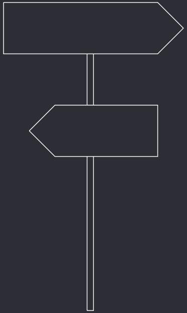


Conclusion

- Contract generation is generally possible
- Few test cases result in a relatively accurate contract, however, the contract slowly keeps getting better
- The current contract template limits the precision

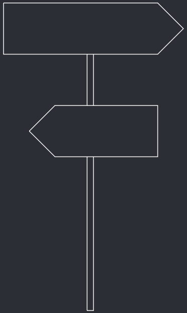


Future Work



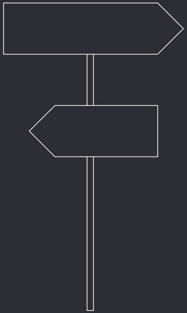
Future Work

- Improved contract templates
  - alignedness of values
  - branch decisions



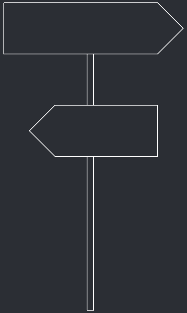
Future Work

- Improved contract templates
  - alignedness of values
  - branch decisions
- Different test case generation methods
  - loop structures
  - load store sequences



Future Work

- Improved contract templates
  - alignedness of values
  - branch decisions
- Different test case generation methods
  - loop structures
  - load store sequences



Thank you for your attention!

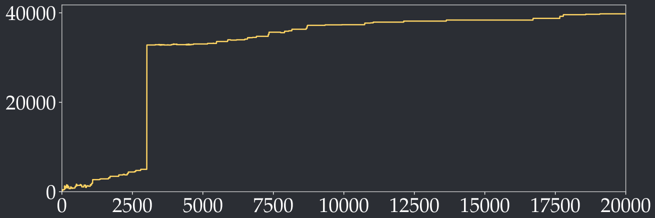
Interesting Numbers

	Size	**Adversary Distinguishable**			
		`Ibex`		`CVA6`	
Training Set	20,000	1421	7.1%	1055	5.2%
Evaluation Set	100,000	7035	7.0%	5573	5.5%

	Ibex	CVA6
Sensitivity	99.64%	97.38%
Precision	31.59%	12.00%
Accuracy	84.80%	57.15%
True Positive	7,010	5,427
False Positive	15,178	39,776
True Negative	77,787	51,724
False Negative	25	146

False Positives on the CVA6 Core

	Size	Adversary Distinguishable			
		Ibex		CVA6	
Training Set	20,000	1421	7.1%	1055	5.2%
Evaluation Set	100,000	7035	7.0%	5573	5.5%



```
r27 ← LB mem[r31 + 3792]
SW mem[r6 + 3169] ← r27
```

```
r27 ← LB mem[r31 + 3792]
SW mem[r6 + 3169] ← r23
```

Only possible observation: `SW: RS2`

Test Case Generation

Each test case has two programs - one for each core.

Test Case Generation

Each test case has two programs - one for each core.

- Create a random initial architectural state

Test Case Generation

Each test case has two programs - one for each core.

- Create a random initial architectural state
- Generate a random instruction

Test Case Generation

Each test case has two programs - one for each core.

- Create a random initial architectural state
- Generate a random instruction
- Derive two similar programs from this instruction

Test Case Generation

Each test case has two programs - one for each core.

- Create a random initial architectural state
- Generate a random instruction
- Derive two similar programs from this instruction

Example:

Generated instruction:	`add rd ← rs1 + rs2`
Observation:	`REG_RS1`

Test Case Generation

Each test case has two programs - one for each core.

- Create a random initial architectural state
- Generate a random instruction
- Derive two similar programs from this instruction

Example:

Generated instruction: `add rd ← rs1 + rs2`

Observation: `REG_RS1`

```
addi rs1 ← r0  + x                addi rs1 ← r0  + y
add  rd  ← rs1 + rs2              add  rd  ← rs1 + rs2
```

Contract Computation

Contract Computation

Defining p:

Contract Computation

Defining p:

$$p(\mathtt{CTR}) = |\{(\mu, \mu') \in M_{\mathtt{TC}} \mid \mathtt{CTR}(\mathtt{ARCH}(\mu)) \neq \mathtt{CTR}(\mathtt{ARCH}(\mu'))\}|$$

Contract Computation

Defining p:

$$p(\mathsf{CTR}) = \left|\left\{(\mu, \mu') \in M_{\mathsf{TC}} \mid \mathsf{CTR}(\mathsf{ARCH}(\mu)) \neq \mathsf{CTR}(\mathsf{ARCH}(\mu'))\right\}\right|$$

Formalizing contract computation:

Contract Computation

Defining p:

$$p(\mathtt{CTR}) = |\{(\mu, \mu') \in M_{\mathtt{TC}} \mid \mathtt{CTR}(\mathtt{ARCH}(\mu)) \neq \mathtt{CTR}(\mathtt{ARCH}(\mu'))\}|$$

Formalizing contract computation:

$$\forall d \in \text{Dist.} \ (\sum_{o \in \mathtt{obs}(d)} s_o) \geq 1 \qquad \text{Contract Satisfaction}$$

Contract Computation

Defining p:

$$p(\mathsf{CTR}) = \left|\{(\mu, \mu') \in M_{\mathsf{TC}} \mid \mathsf{CTR}(\mathsf{ARCH}(\mu)) \neq \mathsf{CTR}(\mathsf{ARCH}(\mu'))\}\right|$$

Formalizing contract computation:

$$\forall d \in \text{Dist.}\ \Big(\sum_{o \in \mathtt{obs}(d)} s_o\Big) \geq 1 \qquad \text{Contract Satisfaction}$$

$$\forall i \in \text{Indist.}\ \bigvee_{o \in \mathtt{obs}(i)} s_o \Rightarrow c_i \qquad \text{False Positives}$$

Contract Computation

Defining p:

$$p(\mathrm{CTR}) = |\{(\mu, \mu') \in M_{\mathrm{TC}} \mid \mathrm{CTR}(\mathrm{ARCH}(\mu)) \neq \mathrm{CTR}(\mathrm{ARCH}(\mu'))\}|$$

Formalizing contract computation:

$$\forall d \in \text{Dist. } \Big(\sum_{o \in \mathtt{obs}(d)} s_o \Big) \geq 1$$

Contract Satisfaction

$$\forall i \in \text{Indist. } \bigvee_{o \in \mathtt{obs}(i)} s_o \Rightarrow c_i$$

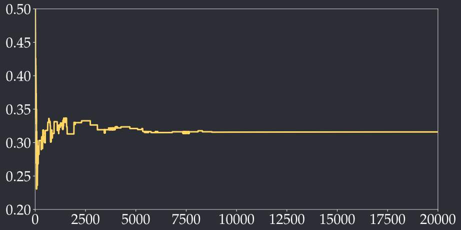
False Positives

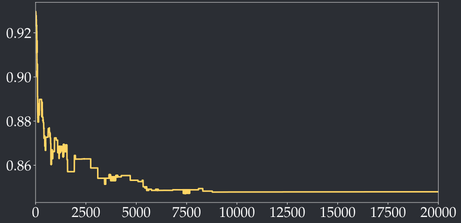
$$\min\Big(\sum_{i \in \text{Indist.}} c_i \Big)$$

Optimize precision

Precision & Accuracy

	Size	Adversary Distinguishable			
		Ibex		CVA6	
Training Set	20,000	1421	7.1%	1055	5.2%
Evaluation Set	100,000	7035	7.0%	5573	5.5%





$$\text{Precision} = \frac{\text{True Positive}}{\text{Predicted Positive}}$$

$$\text{Accuracy} = \frac{\text{Correctly Predicted}}{\text{Total}}$$