

# Static Analysis by Abstract Interpretation for Quantitative Program Properties

Denis Mazzucato  
PhD Defense

10th December 2024



# Static Analysis by Abstract Interpretation for Quantitative Program Properties



**THÈSE DE DOCTORAT**  
DE L'UNIVERSITÉ PSL

Préparée à l'École Normale Supérieure

**Static Analysis by Abstract Interpretation of  
Quantitative Program Properties**

Analyse Statique par Interprétation Abstraite de Propriétés Quantitatives de Programmes

Soutenue par  
**Denis MAZZUCATO**  
Le 10 décembre 2024

École doctorale n°386  
**Sciences Mathématiques de  
Paris Centre**

Spécialité  
**Informatique**

**Composition du jury :**

Thomas JENSEN  
INRIA

*Rapporteur*

Isabella MASTROENI  
Università degli Studi di Verona

*Rapportrice*

Bor-Yuh Evan CHANG  
University of Colorado Boulder & Amazon

*Examinateur*

Bernadette CHARRON-BOST  
CNRS & École Normale Supérieure | PSL

*Examinateuse*

Matthieu MARTEL  
Université de Perpignan

*Examinateur*

Antoine MINÉ  
Sorbonne Université

*Examinateur*

Corina S. PĂSĂREANU  
Carnegie Mellon University & NASA Ames

*Examinateuse*

Caterina URBAN  
INRIA & École Normale Supérieure | PSL

*Directrice de thèse*

# Program Properties



## Static Analysis by Abstract Interpretation of Quantitative Program Properties

Analyse Statique par Interprétation Abstraite de Propriétés Quantitatives de Programmes

Soutenue par  
**Denis MAZZUCATO**  
Le 10 décembre 2024

École doctorale n°386  
**Sciences Mathématiques de  
Paris Centre**

Spécialité  
**Informatique**

Composition du jury :	
Thomas JENSEN INRIA	Rapporteur
Isabella MASTROENI Université de Paris	Rapporteur
Bor-Yuh E. CHOU University of Cambridge	Membre
Bernadette BERTRAN CNRS & École Normale Supérieure	Membre
Matthieu LEHRER Université de Paris	Membre
Antoine MINOFARIAN Sorbonne Université	Membre
Corina S. SEGUIN Carnegie Mellon University	Membre
Caterina TRONCI INRIA & École Normale Supérieure	Membre

# Program Properties

## Introduction | 1

In today's world, software rules safety-critical systems such as nuclear power plants [1], car engines [2], airplane control systems [3], and medical devices [4]. The reliability of these systems is based on their correctness. Any failure or vulnerability in safety-critical software poses significant risks, potentially causing financial losses or threatening human safety. Recent advances in software development have led to increasingly complex software systems. As software grows in complexity, the likelihood

1.1	Software Quality . . . . .	3
1.2	Input Data Usage . . . . .	5
1.3	Quantitative Properties . . . . .	6
1.3.1	Extensional Properties . . . . .	7
1.3.2	Intensional Properties . . . . .	8

[1]: Krasner (2021), 'The cost of poor soft-

## Static Analysis by Abstract Interpretation of Quantitative Program Properties

Analyse Statique par Interprétation Abstraite de Propriétés Quantitatives de Programmes

Soutenue par  
**Denis MAZZUCATO**  
Le 10 décembre 2024

École doctorale n°386  
**Sciences Mathématiques de  
Paris Centre**

Spécialité  
**Informatique**

Composition du jury :	
Thomas JENSEN INRIA	Rapporteur
Isabella MASTROENI Université de Paris	Rapporteur
Bor-Yuh E. CHOU University of Cambridge	Membre
Bernadette BERTRAN CNRS & École Normale Supérieure	Membre
Matthieu LEHRER Université de Paris	Membre
Antoine MINOFARIAN Sorbonne Université	Membre
Corina S. SEGUIN Carnegie Mellon University	Membre
Caterina TRONCI INRIA & École Normale Supérieure	Membre

# Program Properties

## Introduction | 1

In today's world, software rules safety-critical systems such as nuclear power plants [1], car engines [2], airplane control systems [3], and medical devices [4]. The reliability of these systems is based on their correctness. Any failure or vulnerability in safety-critical software poses significant risks, potentially causing financial losses or threatening human safety. Recent advances in software development have led to increasingly complex software systems. As software grows in complexity, the likelihood

1.1	Software Quality . . . . .	3
1.2	Input Data Usage . . . . .	5
1.3	Quantitative Properties . . . . .	6
1.3.1	Extensional Properties . . . . .	7
1.3.2	Intensional Properties . . . . .	8

[1]: Krasner (2021), 'The cost of poor soft-

## Static Analysis by Abstract Interpretation of Quantitative Program Properties

Analyse Statique par Interprétation Abstraite de Propriétés Quantitatives de Programmes

Soutenue par  
**Denis MAZZUCATO**  
Le 10 décembre 2024

École doctorale n°386  
**Sciences Mathématiques de Paris Centre**

Spécialité  
**Informatique**

- Safety
- Security
- Performance
- ...

# Program Properties

## Introduction | 1

In today's world, software rules safety-critical systems such as nuclear power plants [1], car engines [2], airplane control systems [3], and medical devices [4]. The reliability of these systems is based on their correctness. Any failure or vulnerability in safety-critical software poses significant risks, potentially causing financial losses or threatening human safety. Recent advances in software development have led to increasingly complex software systems. As software grows in complexity, the likelihood

1.1	Software Quality . . . . .	3
1.2	Input Data Usage . . . . .	5
1.3	Quantitative Properties . . . . .	6
1.3.1	Extensional Properties . . . . .	7
1.3.2	Intensional Properties . . . . .	8

[1]: Krasner (2021), 'The cost of poor soft-

**Software Reliability** before deployment is critical to ensure correctness for safety-critical systems

# Program Properties

## Introduction | 1

In today's world, software rules safety-critical systems such as nuclear power plants [1], car engines [2], airplane control systems [3], and medical devices [4]. The reliability of these systems is based on their correctness. Any failure or vulnerability in safety-critical software poses significant risks, potentially causing financial losses or threatening human safety. Recent advances in software development have led to increasingly complex software systems. As software grows in complexity, the likelihood

1.1	Software Quality . . . . .	3
1.2	Input Data Usage . . . . .	5
1.3	Quantitative Properties .	6
1.3.1	Extensional Properties . .	7
1.3.2	Intensional Properties . .	8

[1]: Krasner (2021), 'The cost of poor soft-

# How to ensure Software Quality?

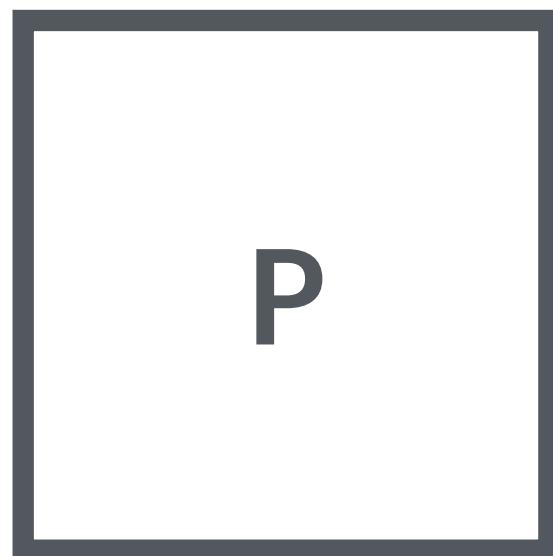
---

# Testing

---

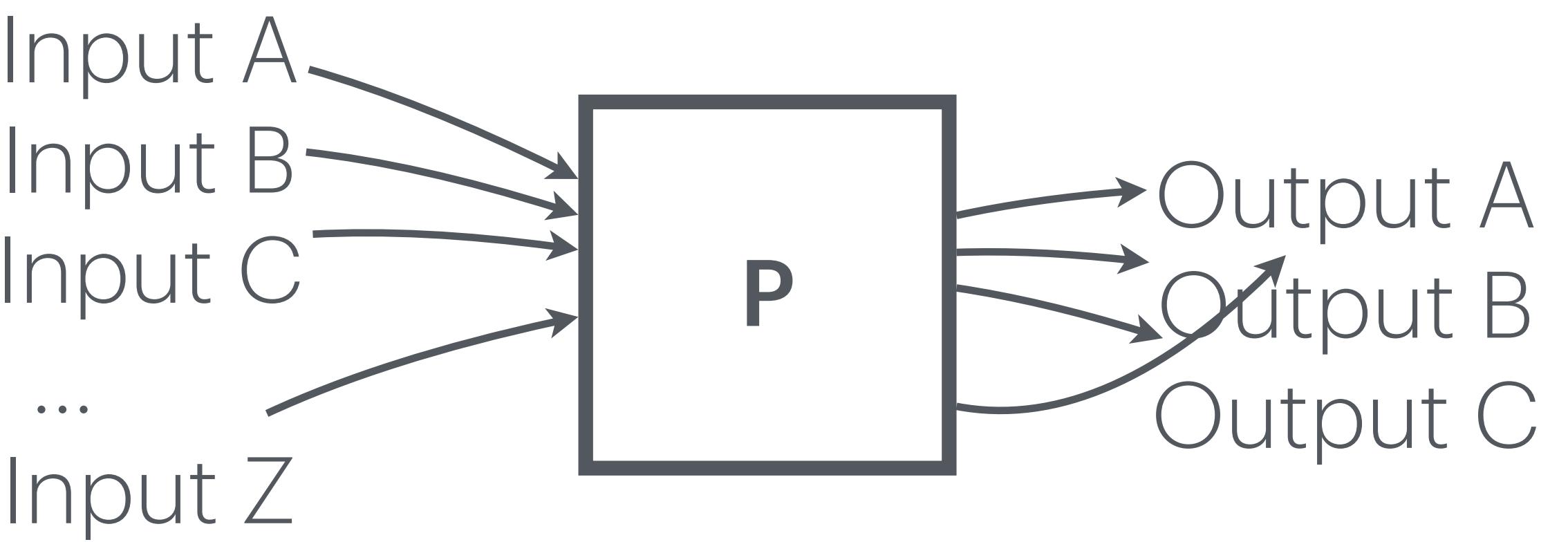
# Testing

---



# Testing

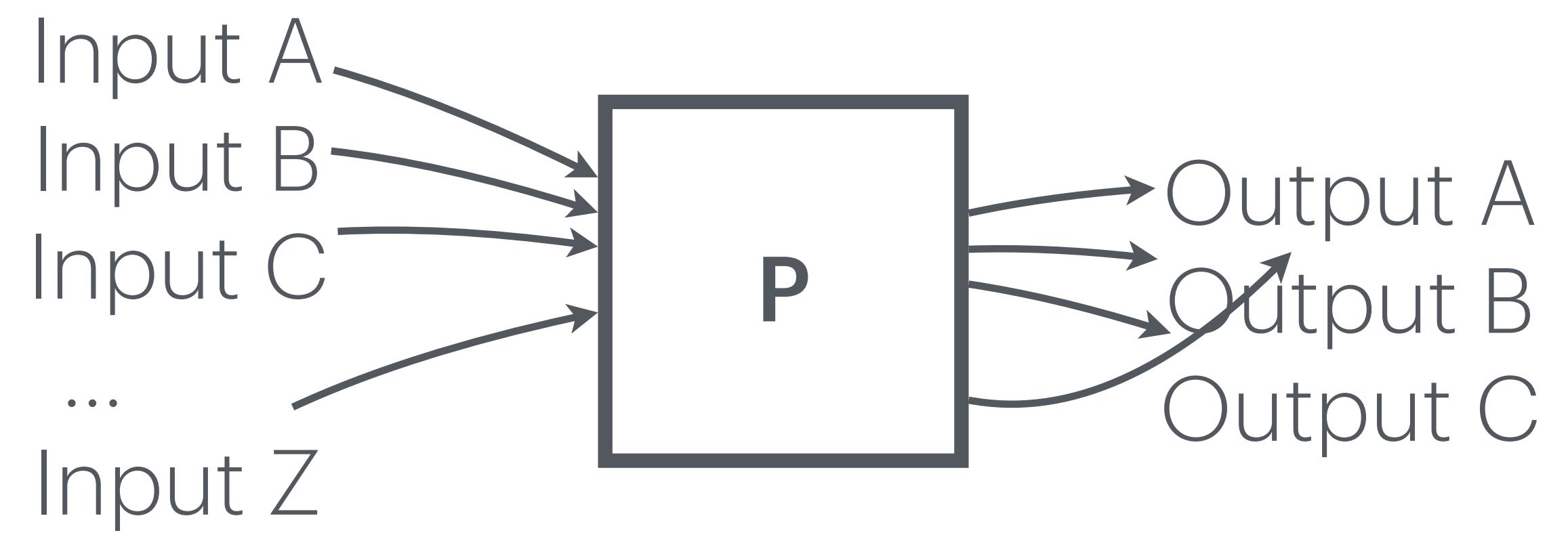
---



# Testing

---

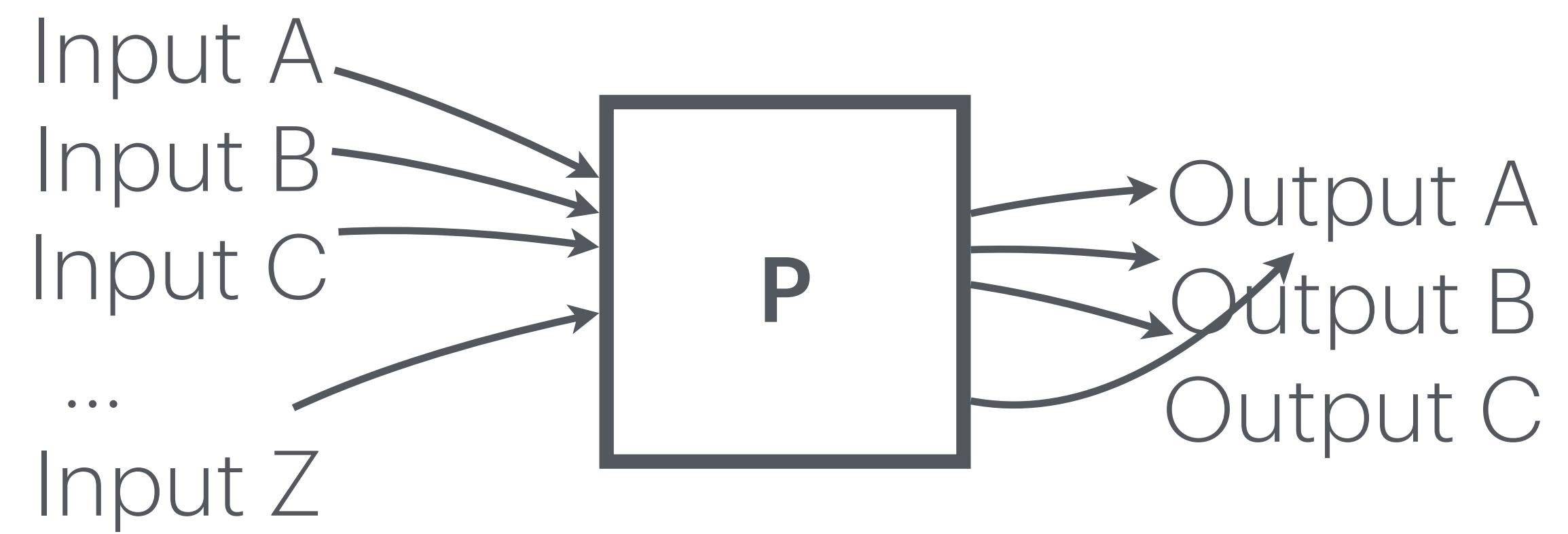
- Most common
- Correct behavior tested empirically



# Testing

---

- Most common
- Correct behavior tested empirically
- Exhaustive testing
- No guarantees on absence of bugs



- Exhaustive testing
- No guarantees on absence of bugs



- Ariane 5 rocket failure  
\$370 million

- Exhaustive testing
- No guarantees on absence of bugs



- Exhaustive testing
- No guarantees on absence of bugs
- Ariane 5 rocket failure  
\$370 million
- Toyota unintended acceleration case  
\$1.2 billion



- Exhaustive testing
- No guarantees on absence of bugs

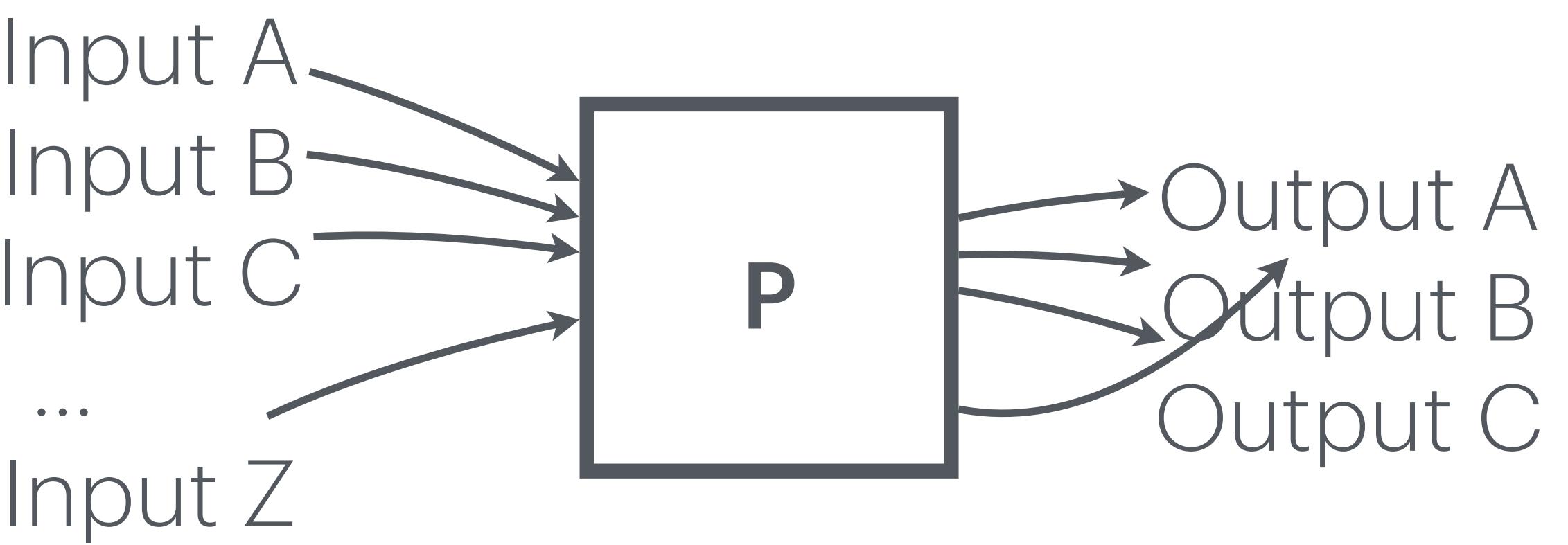
- Ariane 5 rocket failure  
\$370 million
- Toyota unintended acceleration case  
\$1.2 billion



- Exhaustive testing
- No guarantees on absence of bugs
- Ariane 5 rocket failure  
\$370 million
- Toyota unintended acceleration case  
\$1.2 billion
- Therac-25 radiation therapy machine  
9 deaths
- Patriot missile round-off error  
28 deaths
- and many more!  
[https://www.youtube.com/watch?app=desktop&v=lq\\_r7lcNmUk](https://www.youtube.com/watch?app=desktop&v=lq_r7lcNmUk)

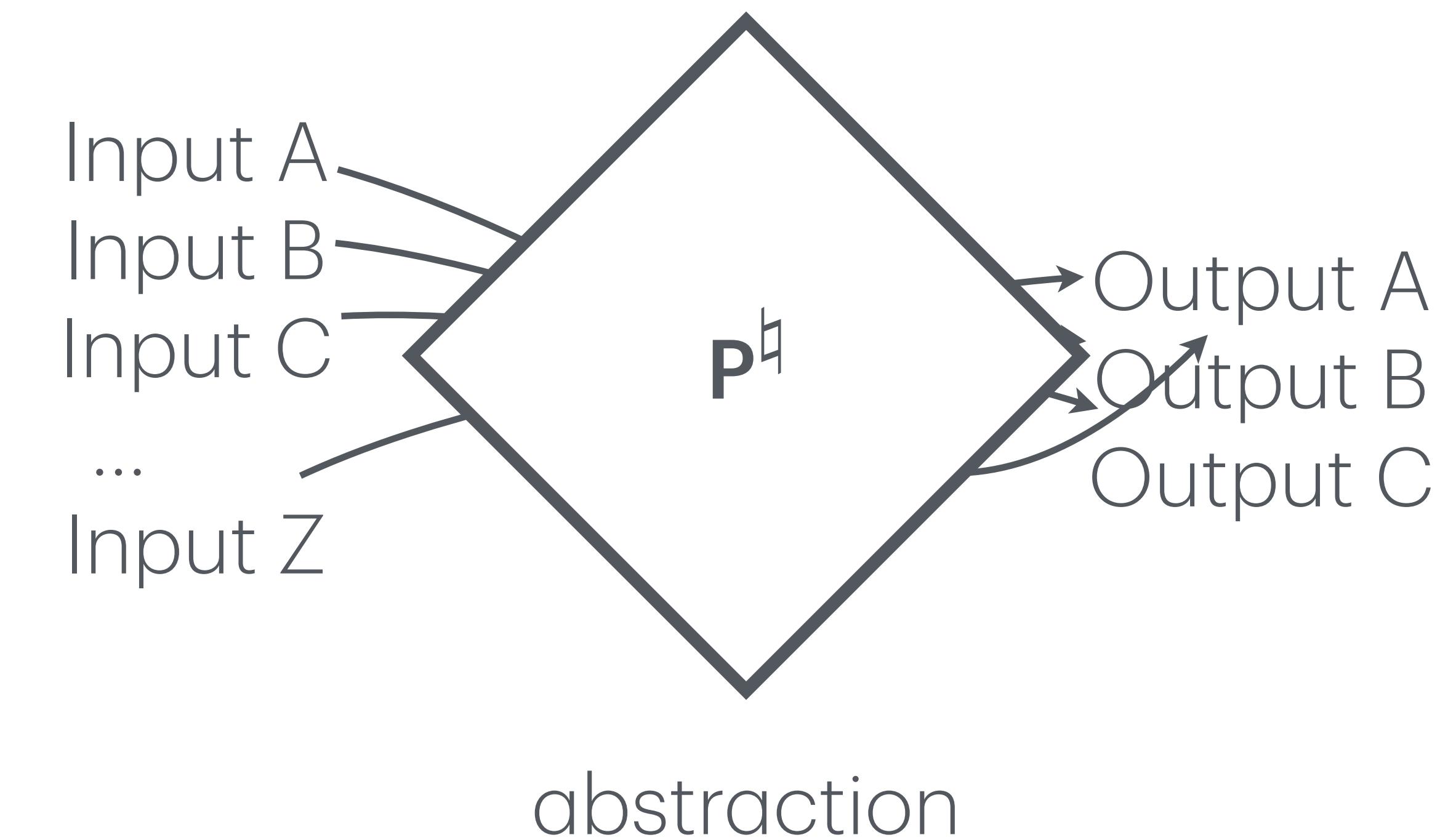
# Formal Methods

---



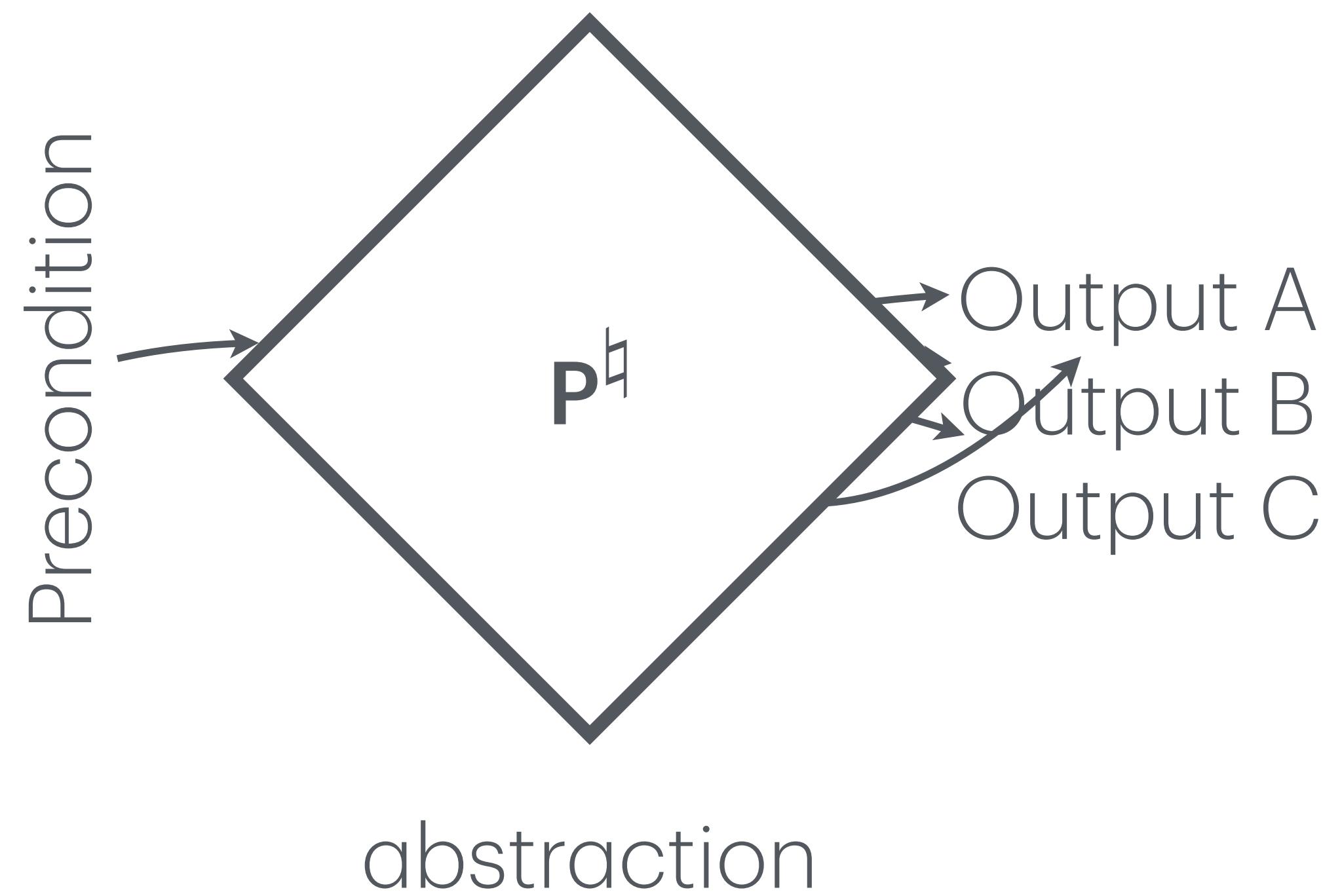
# Formal Methods

---



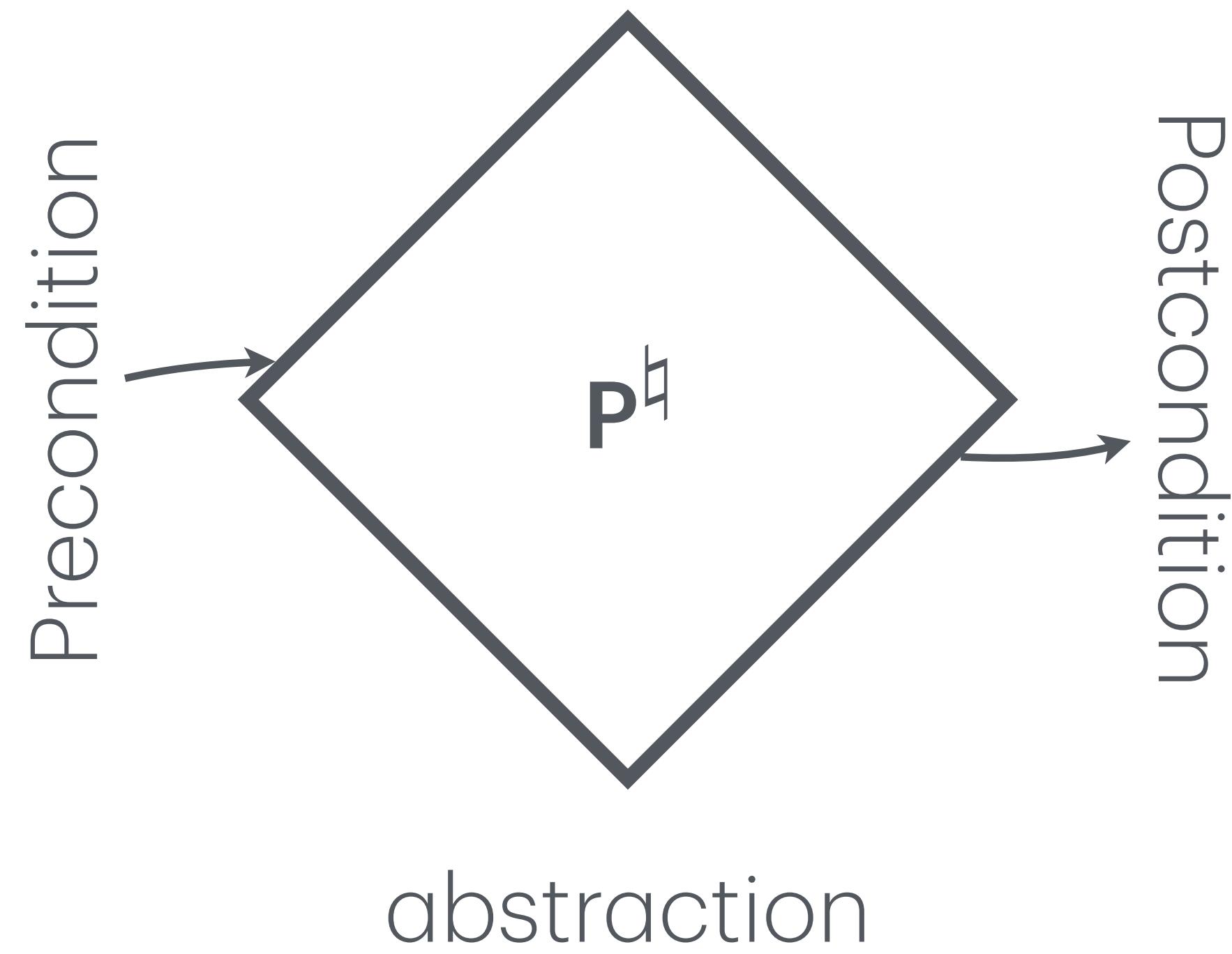
# Formal Methods

---



# Formal Methods

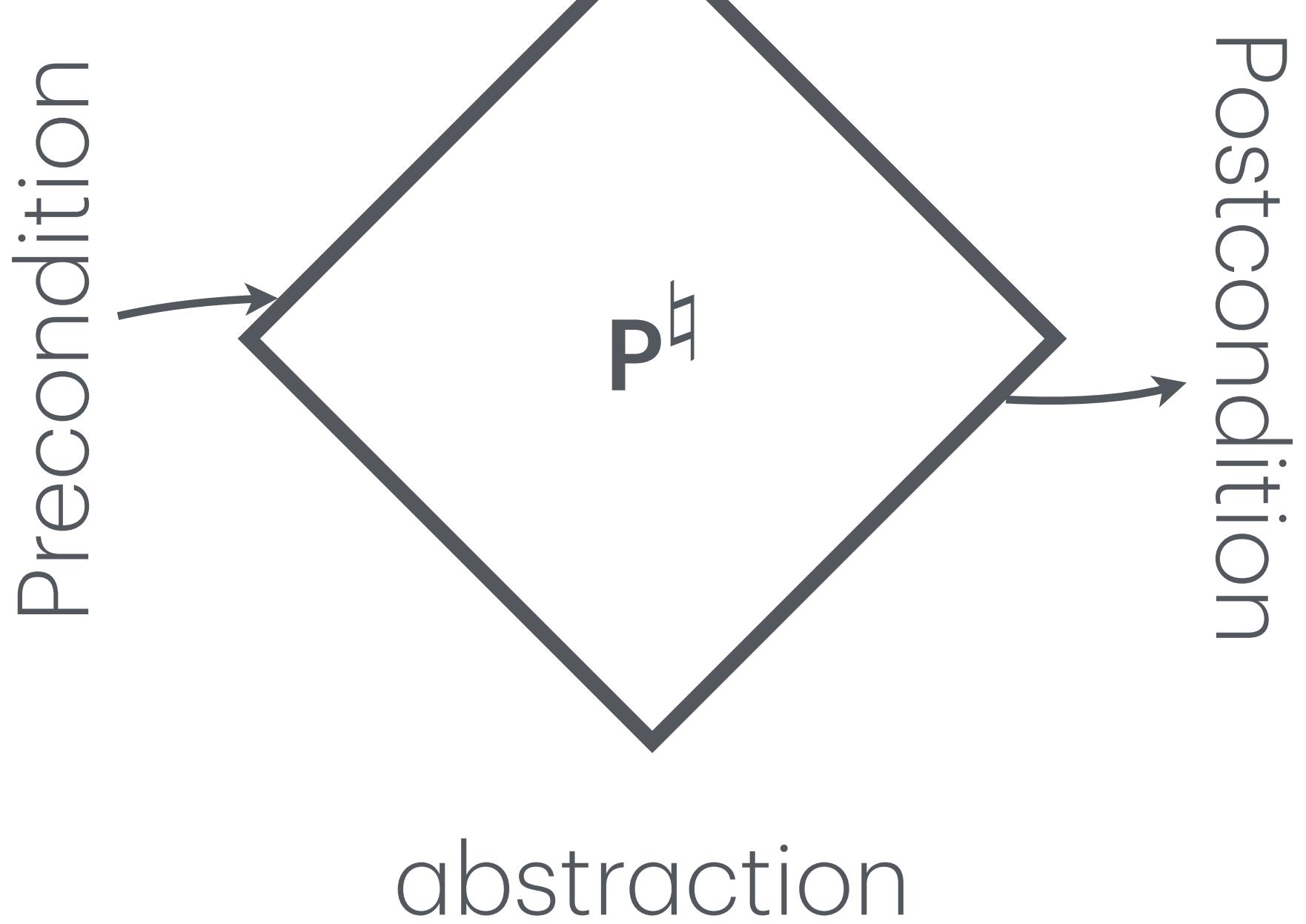
---



# Formal Methods

---

Mathematical Guarantees

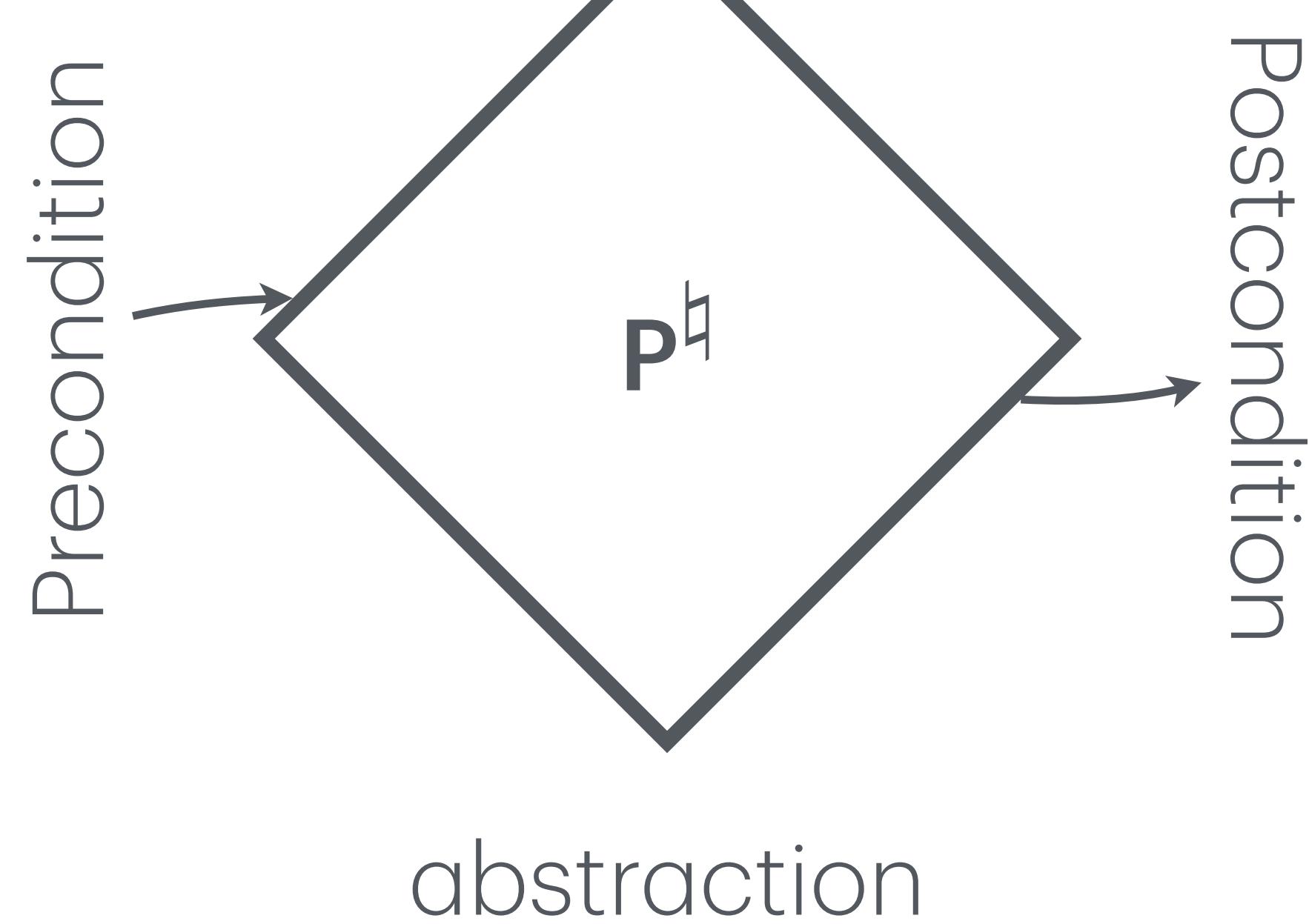


# Formal Methods

---

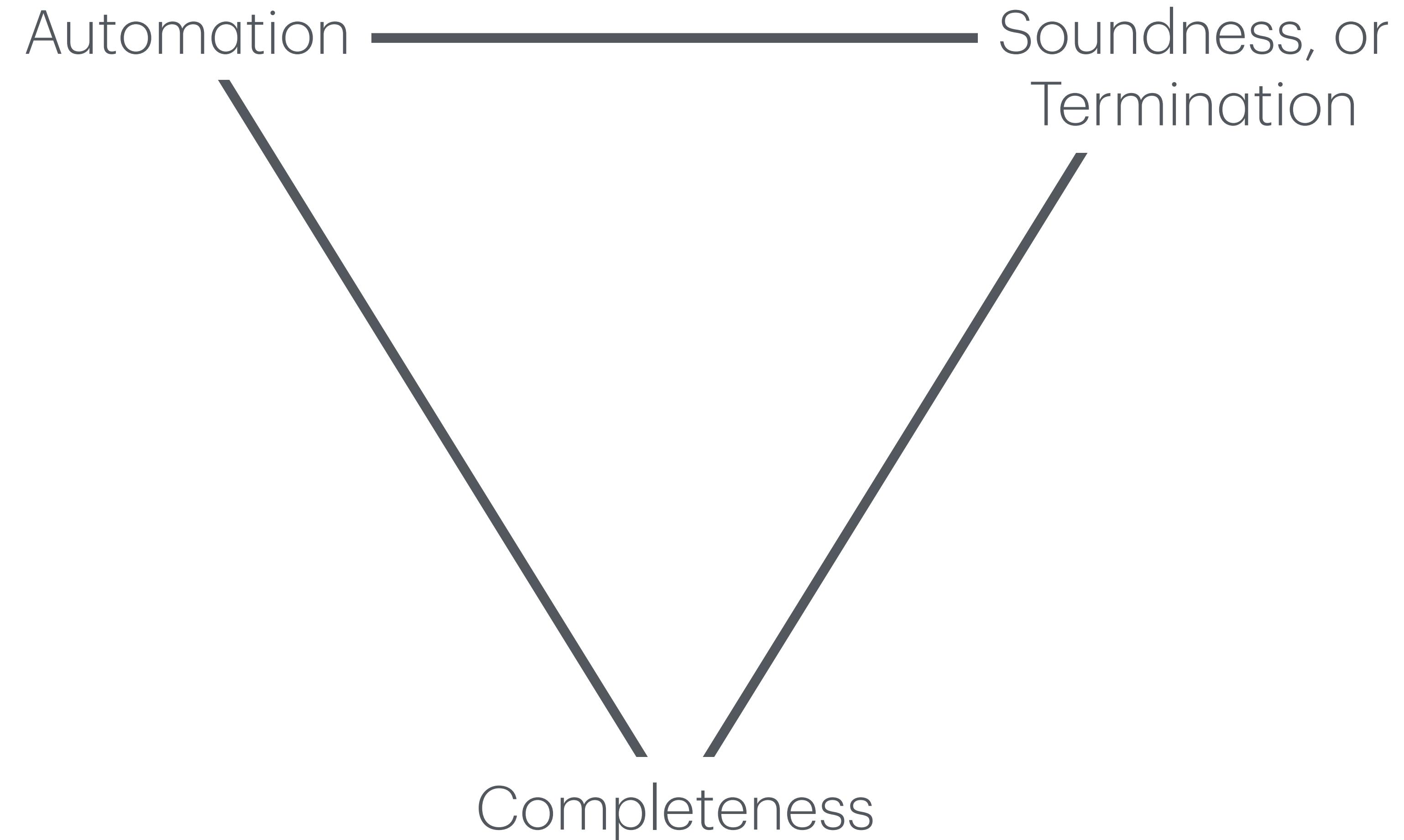
Mathematical Guarantees

Rice's undecidability theorem



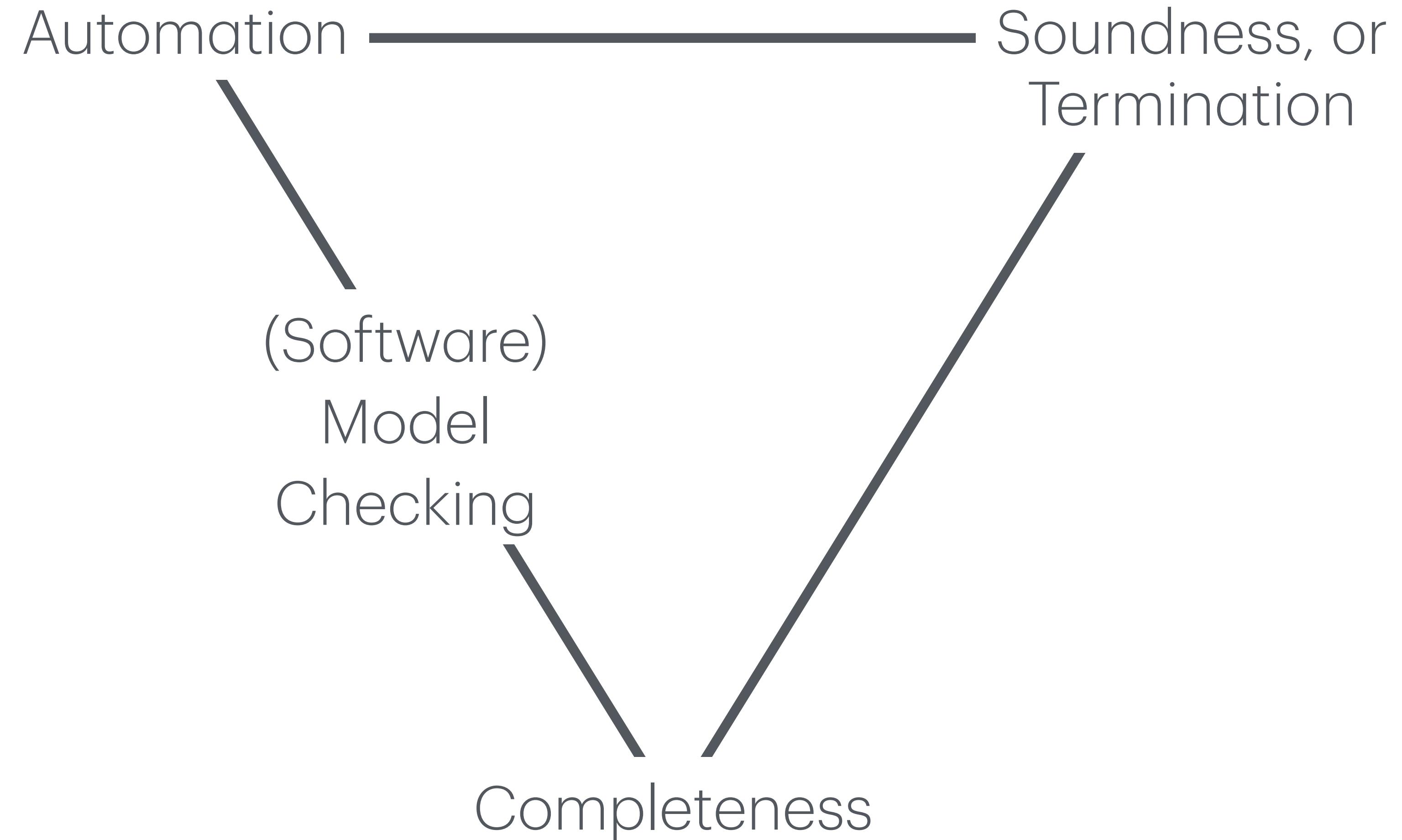
# Formal Methods, Tradeoff:

---



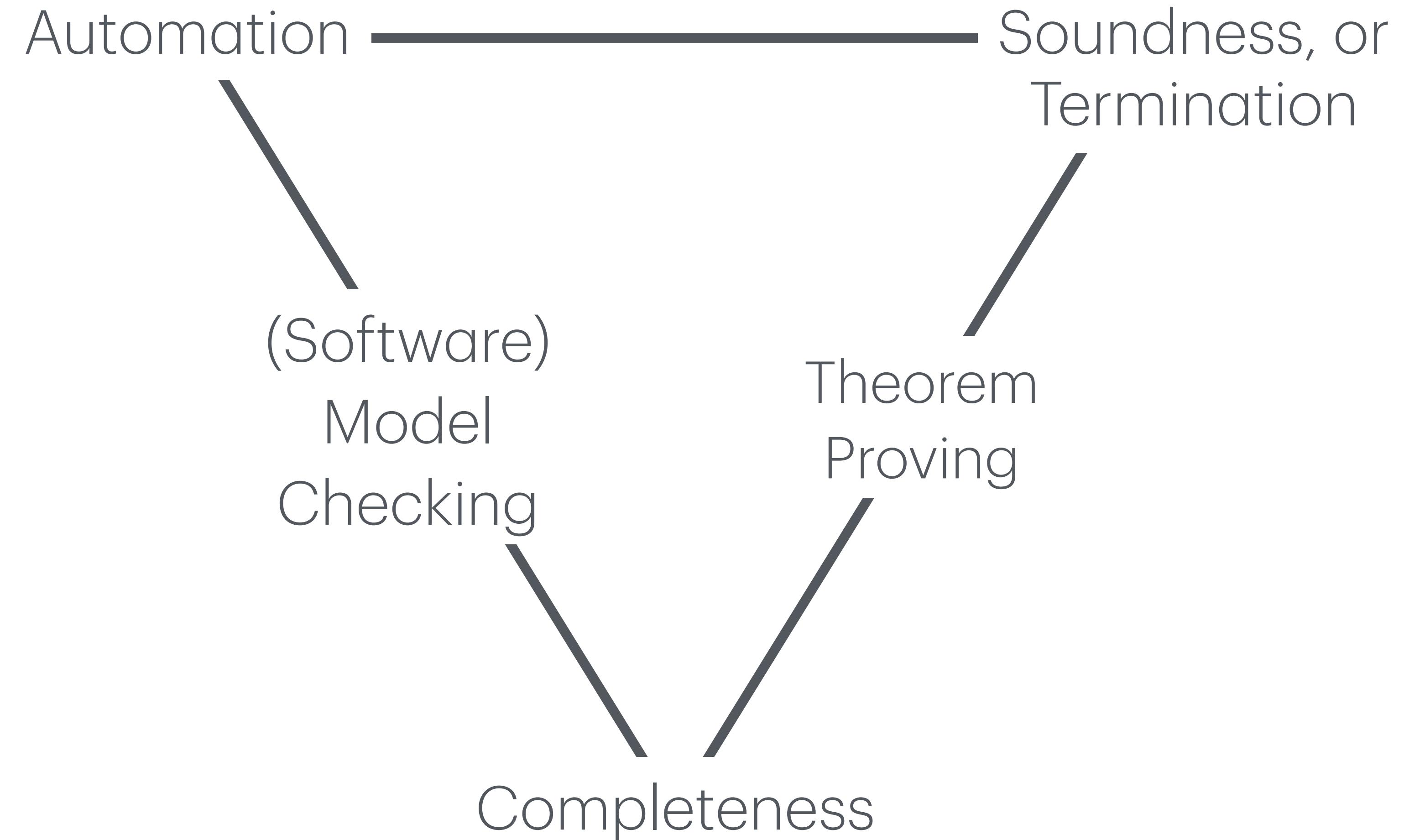
# Formal Methods, Tradeoff:

---



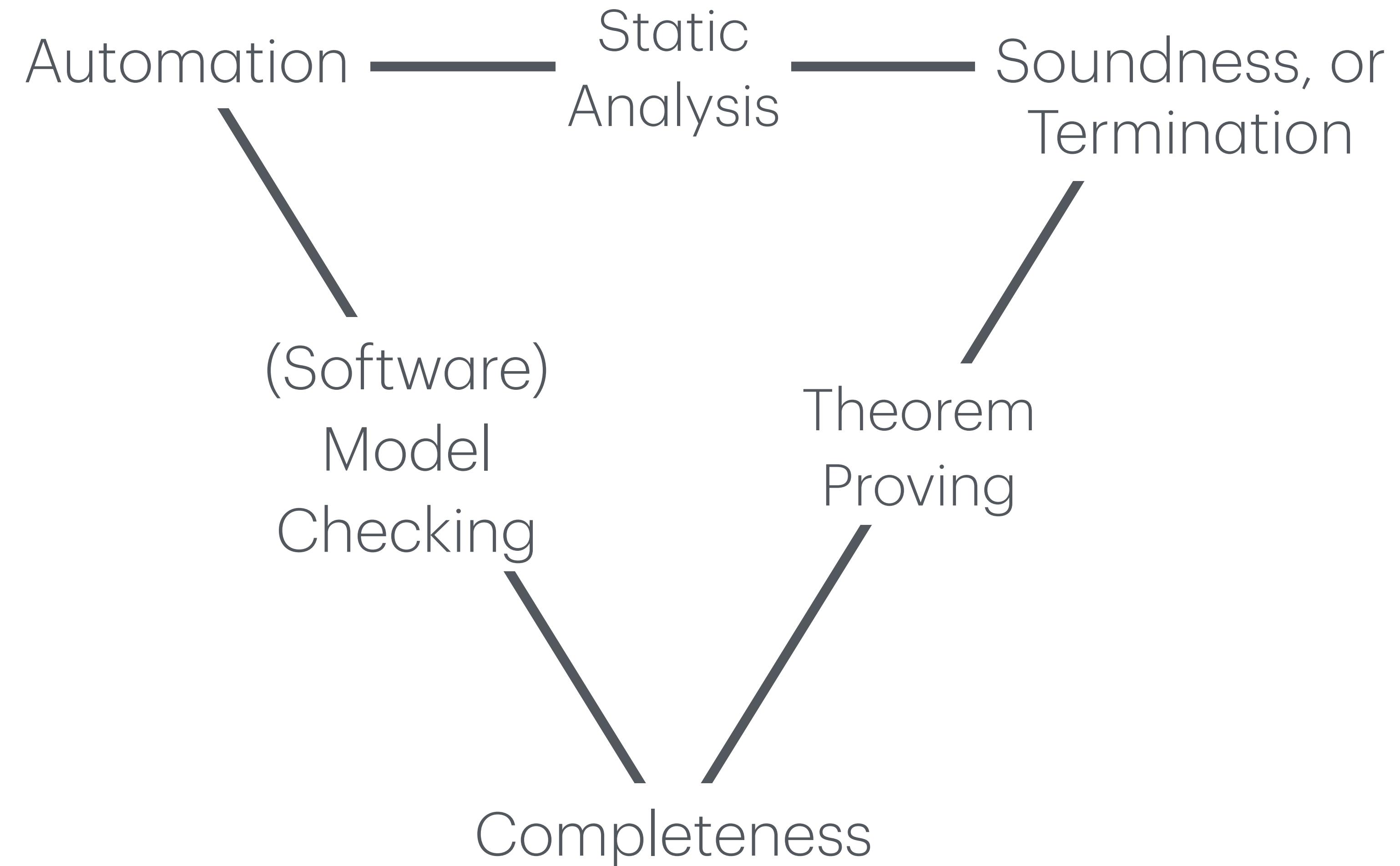
# Formal Methods, Tradeoff:

---



# Formal Methods, Tradeoff:

---



# Static Analysis by Abstract Interpretation for Quantitative Program Properties

# Static Analysis

**Verify** the program source code  
at some **level of abstraction**  
without **user interaction**

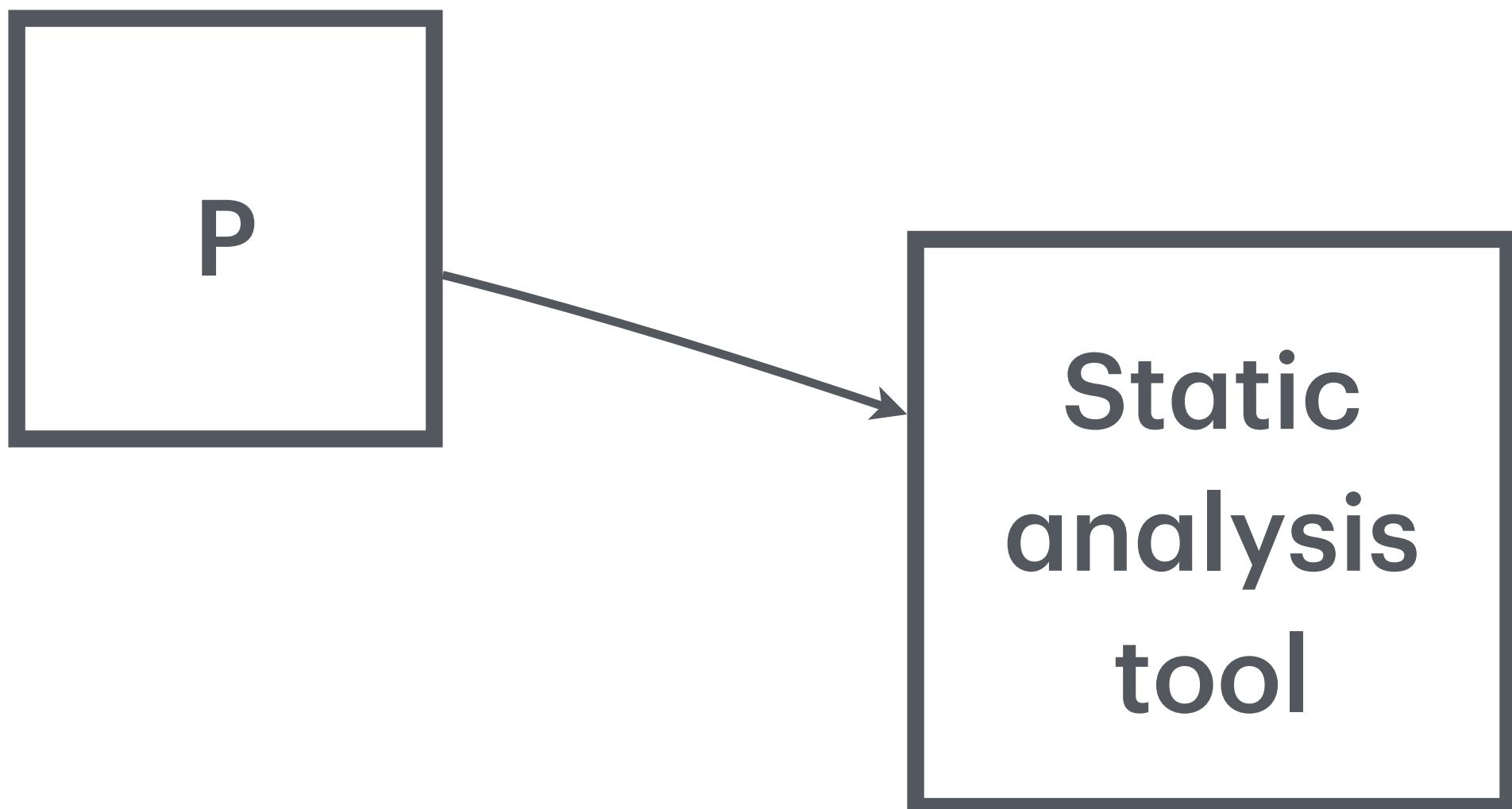
# Static Analysis

**Verify** the program source code  
at some **level of abstraction**  
without **user interaction**

Static  
analysis  
tool

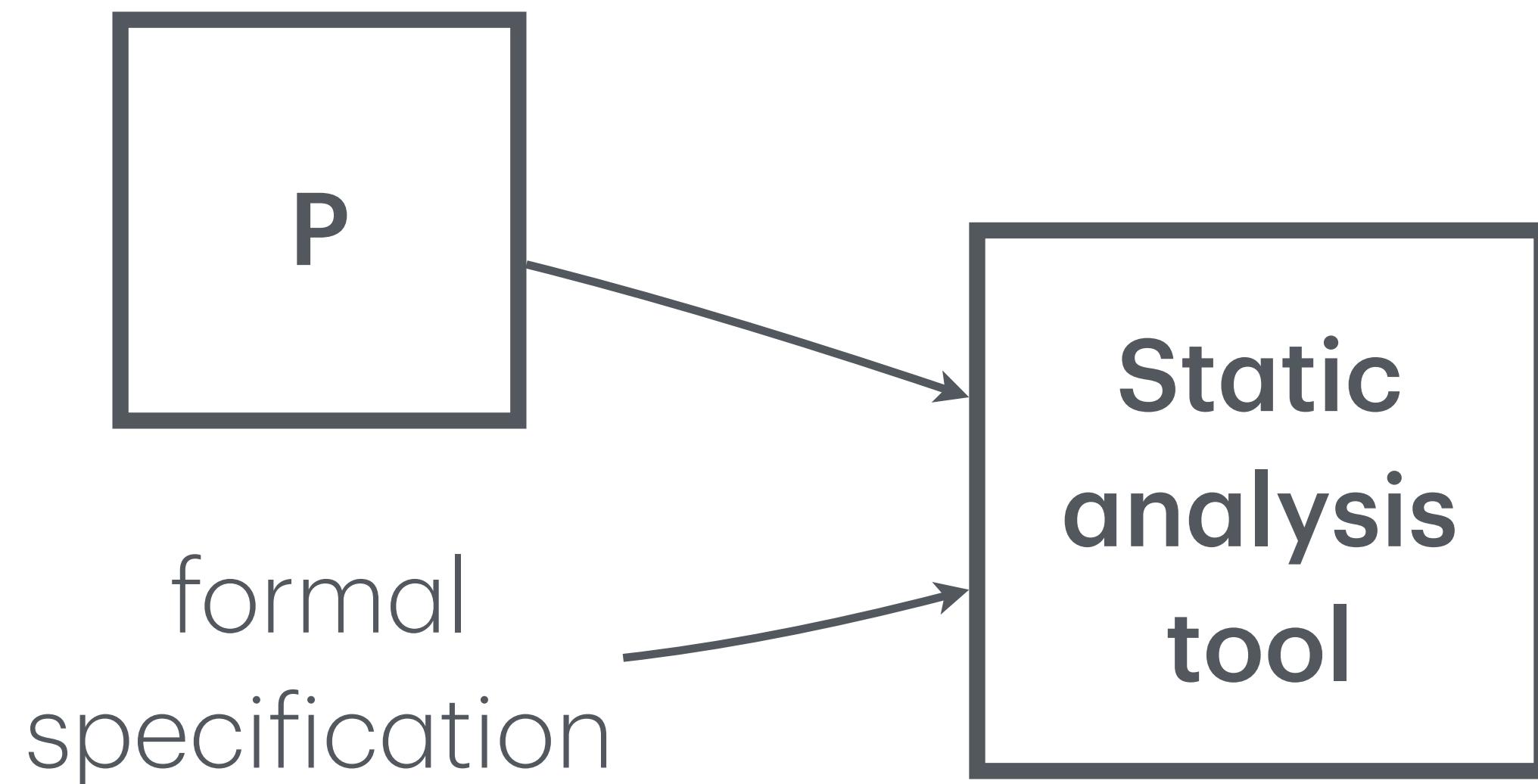
# Static Analysis

**Verify** the program source code  
at some **level of abstraction**  
without **user interaction**



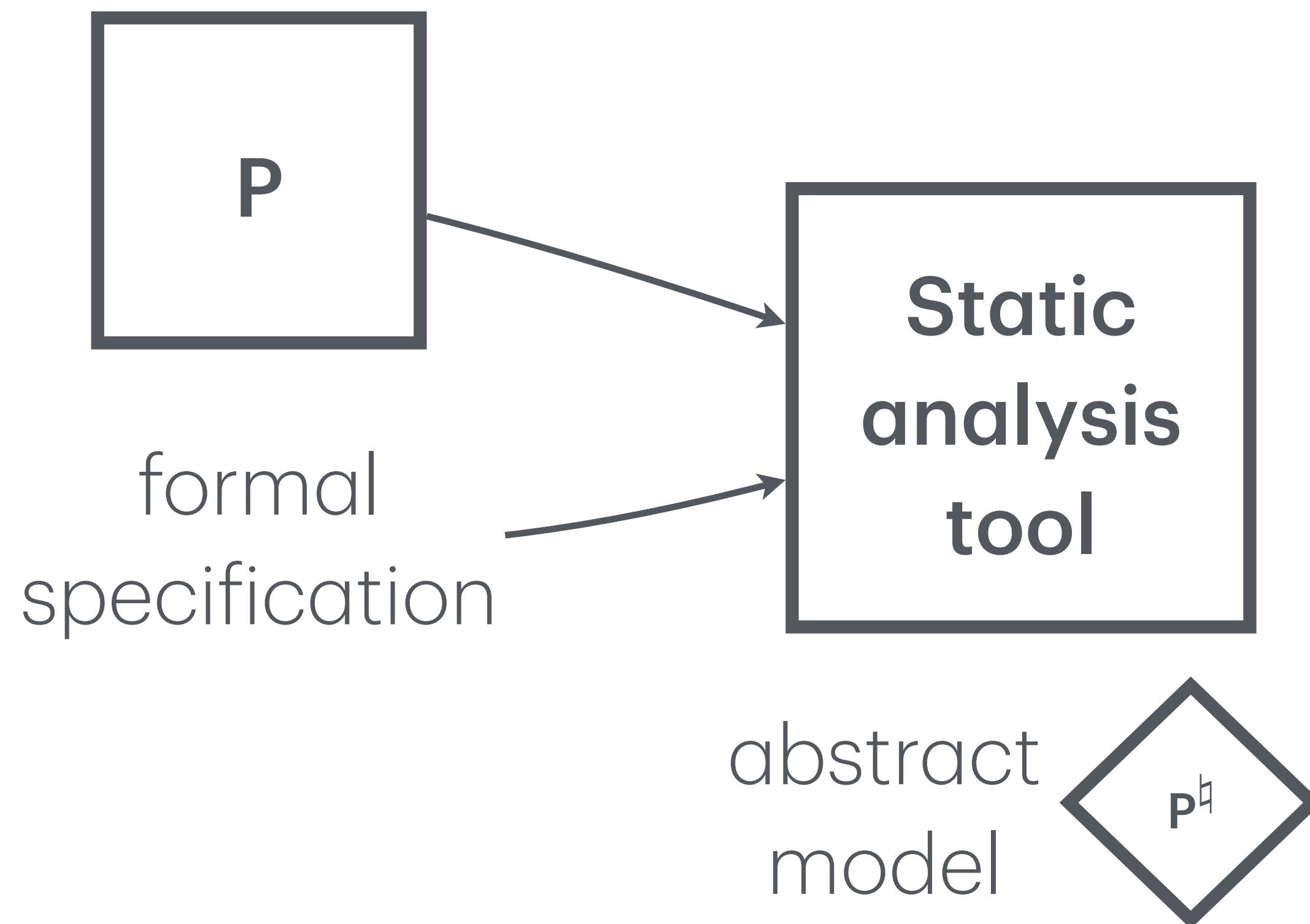
# Static Analysis

**Verify** the program source code  
at some **level of abstraction**  
without **user interaction**



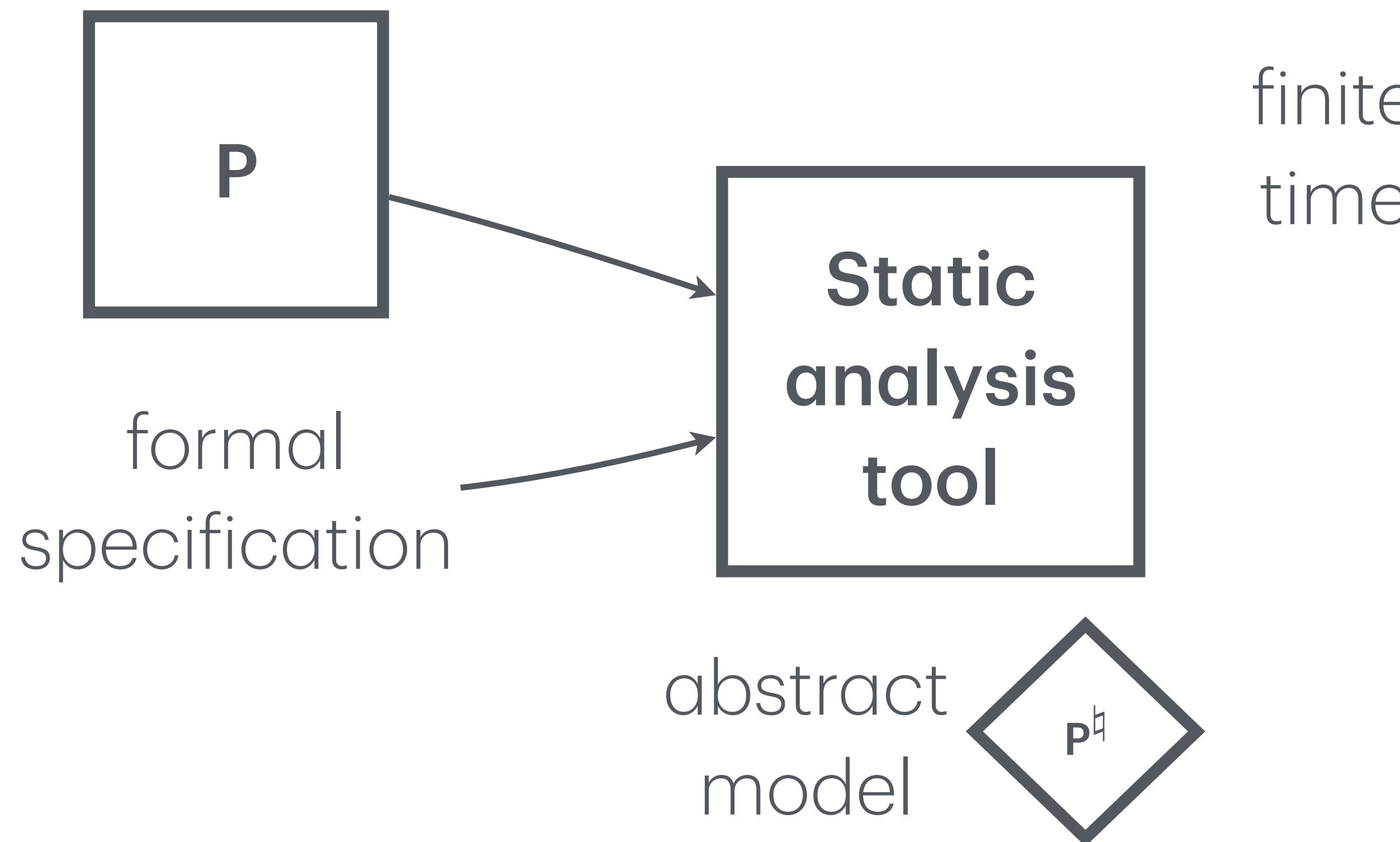
# Static Analysis

**Verify** the program source code  
at some **level of abstraction**  
without **user interaction**



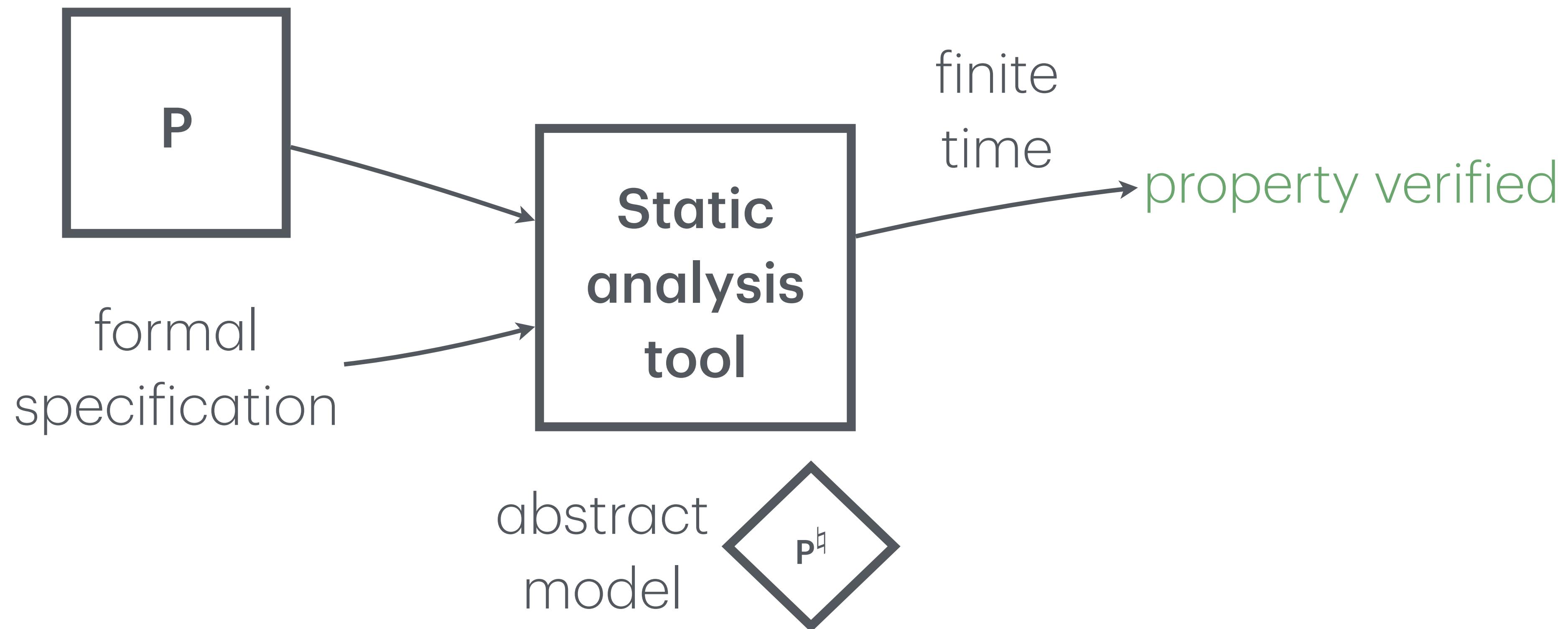
# Static Analysis

**Verify** the program source code  
at some **level of abstraction**  
without **user interaction**



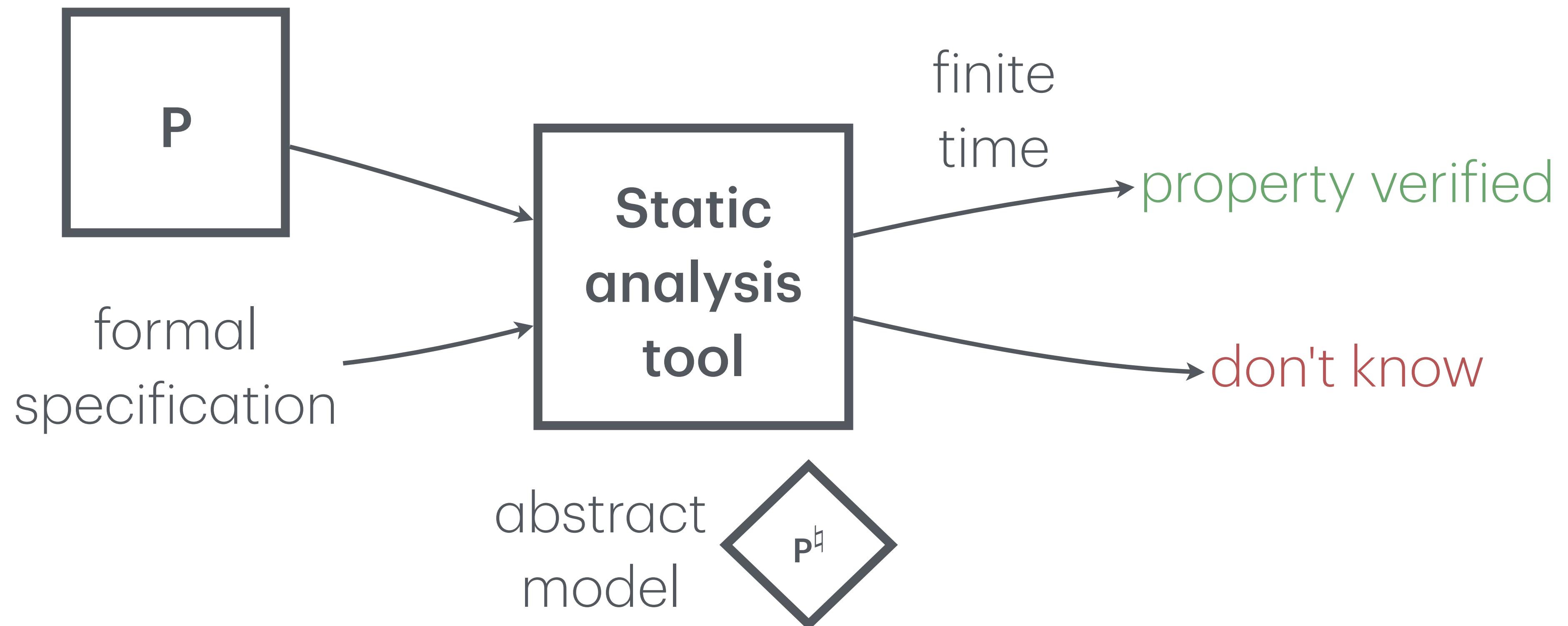
# Static Analysis

**Verify** the program source code  
at some **level of abstraction**  
without **user interaction**



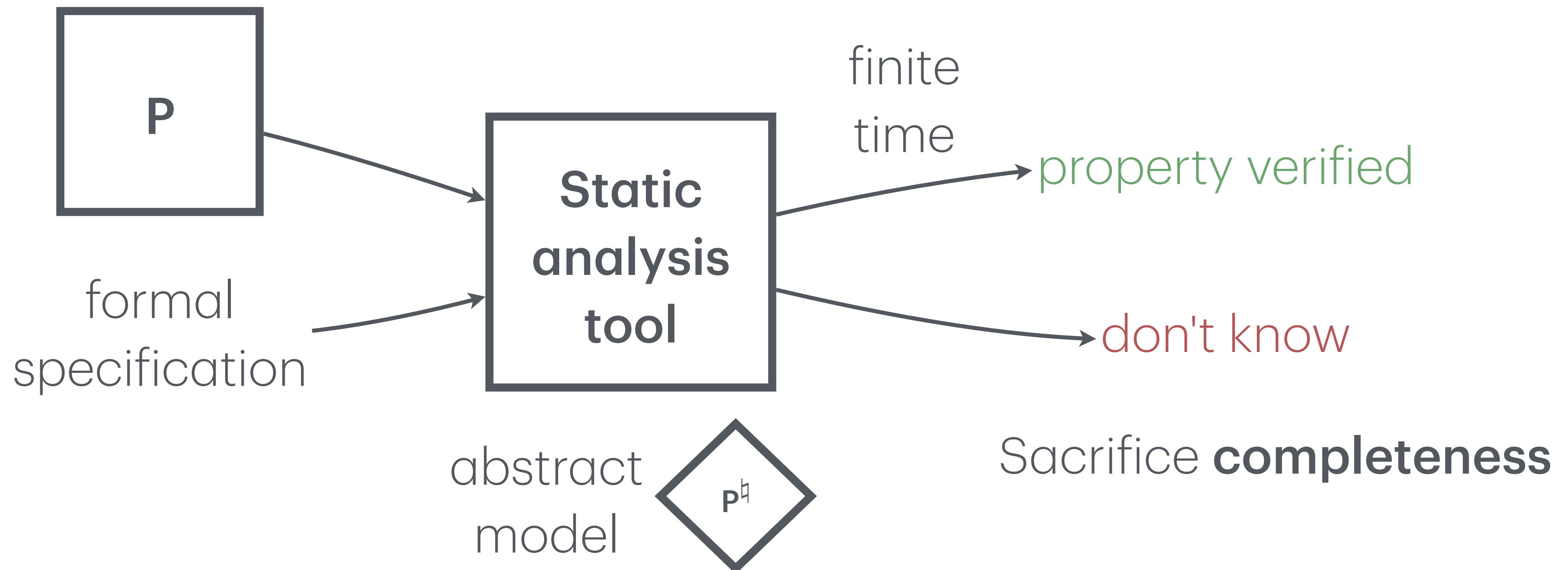
# Static Analysis

**Verify** the program source code  
at some **level of abstraction**  
without **user interaction**



# Static Analysis

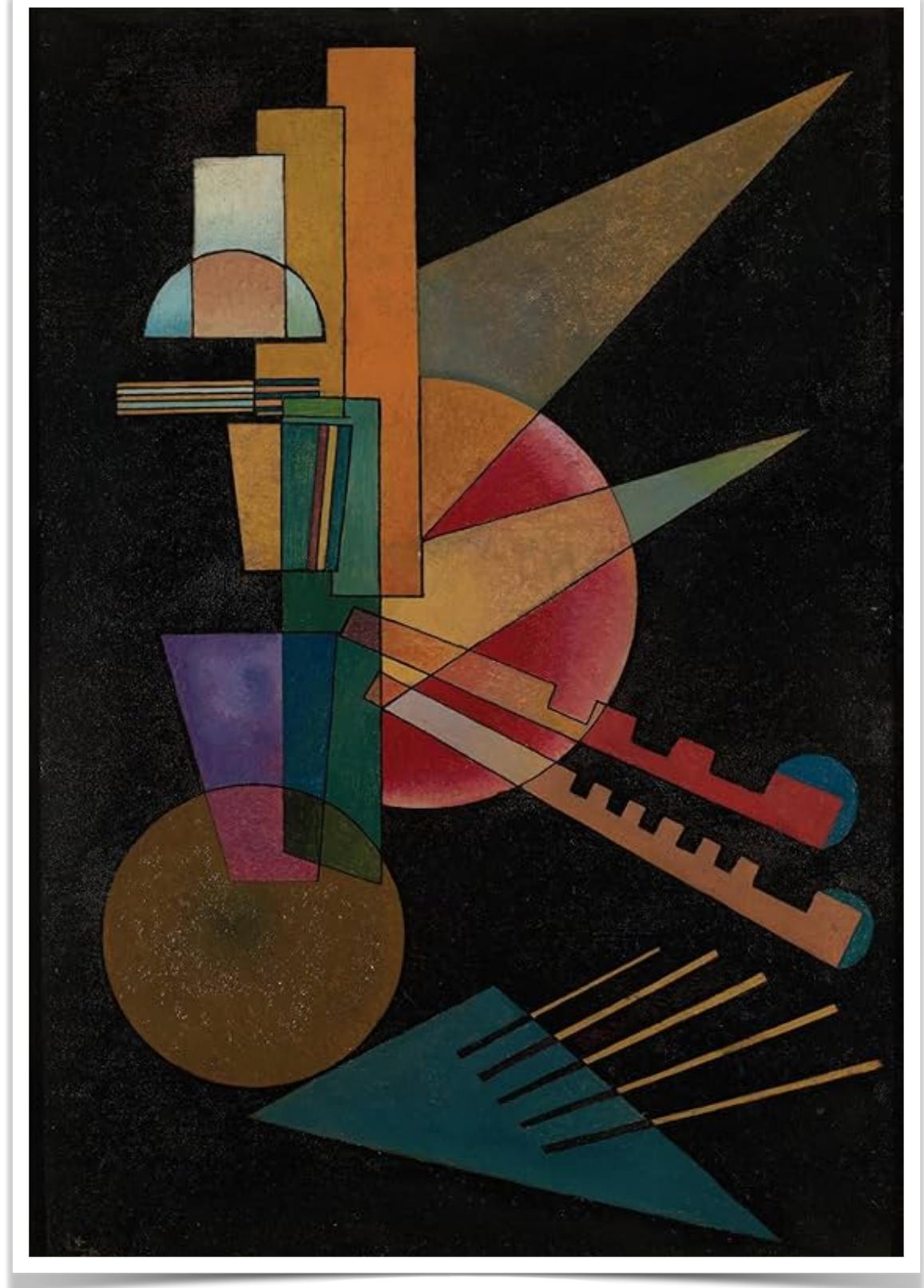
**Verify** the program source code  
at some **level of abstraction**  
without **user interaction**



# Static Analysis by Abstract Interpretation for Quantitative Program Properties

# Abstract Interpretation

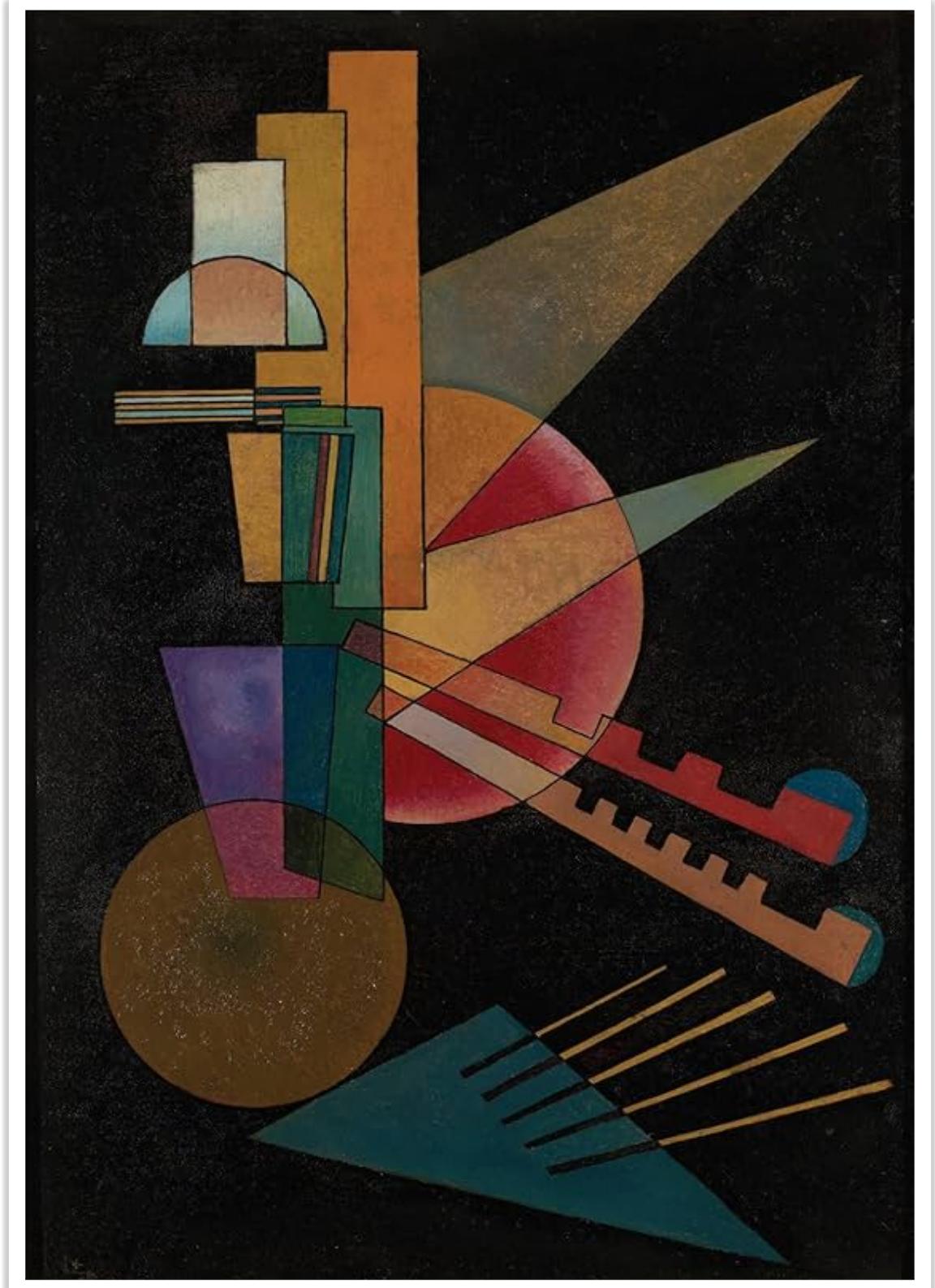
**Approximating** program semantics



# Abstract Interpretation

**Approximating** program semantics

Patrick Cousot and Radhia Cousot



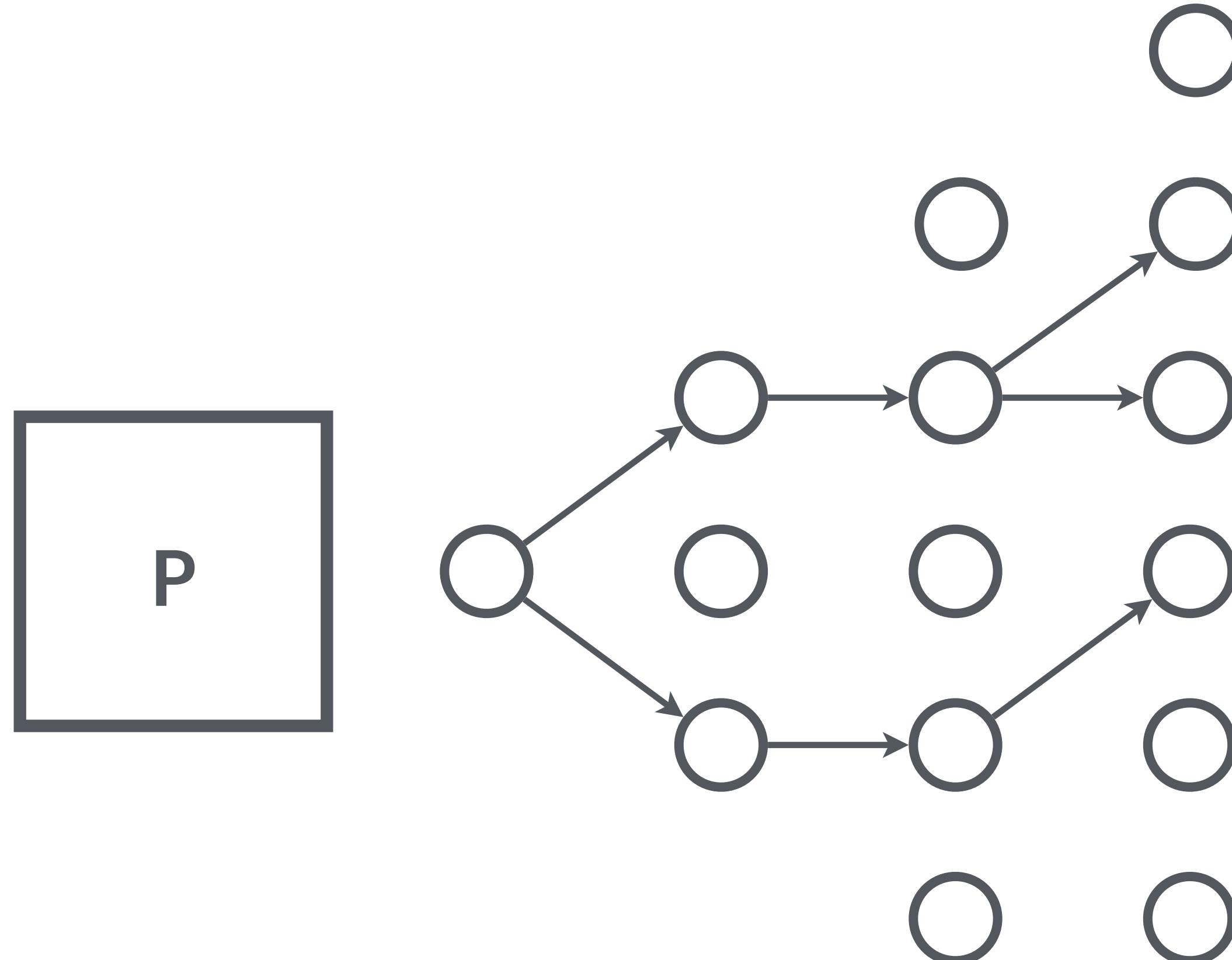
# Abstract interpretation

---



# Abstract interpretation

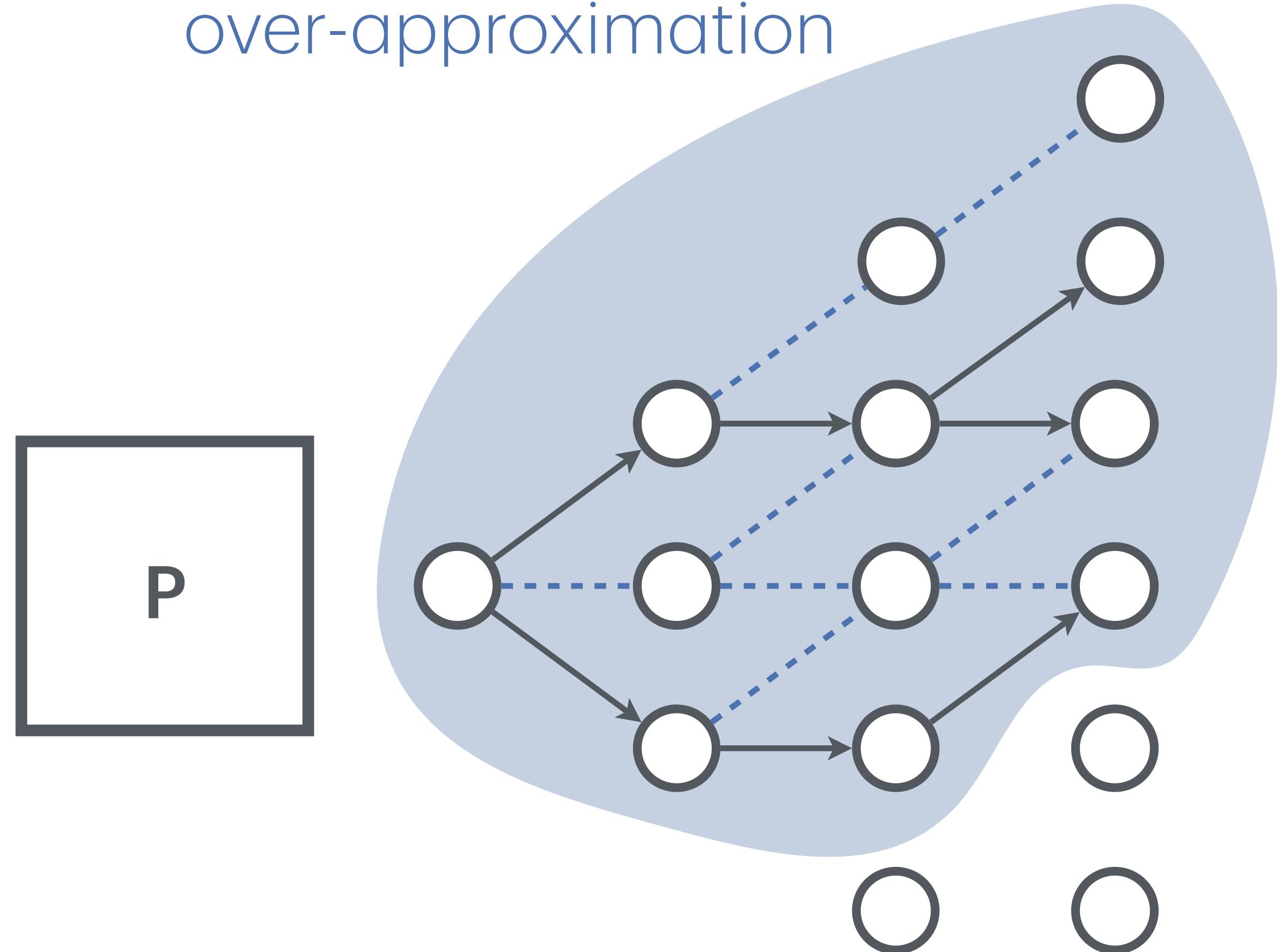
---



trace  
semantics  $\Lambda[P]$

# Abstract interpretation

over-approximation

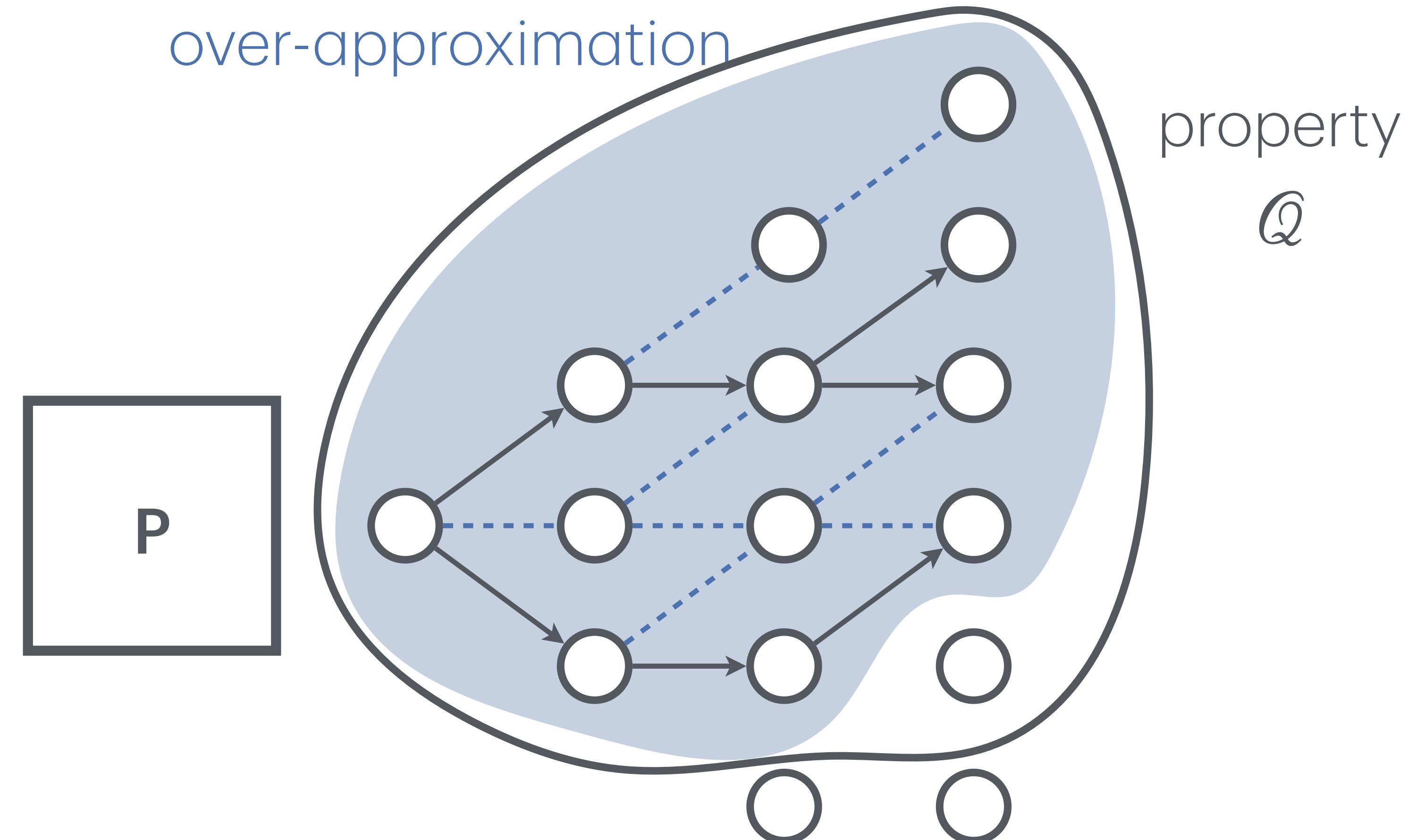


trace  
semantics

$$\Lambda[P] \subseteq \Lambda^\sharp[P]$$

abstract  
semantics

# Abstract interpretation

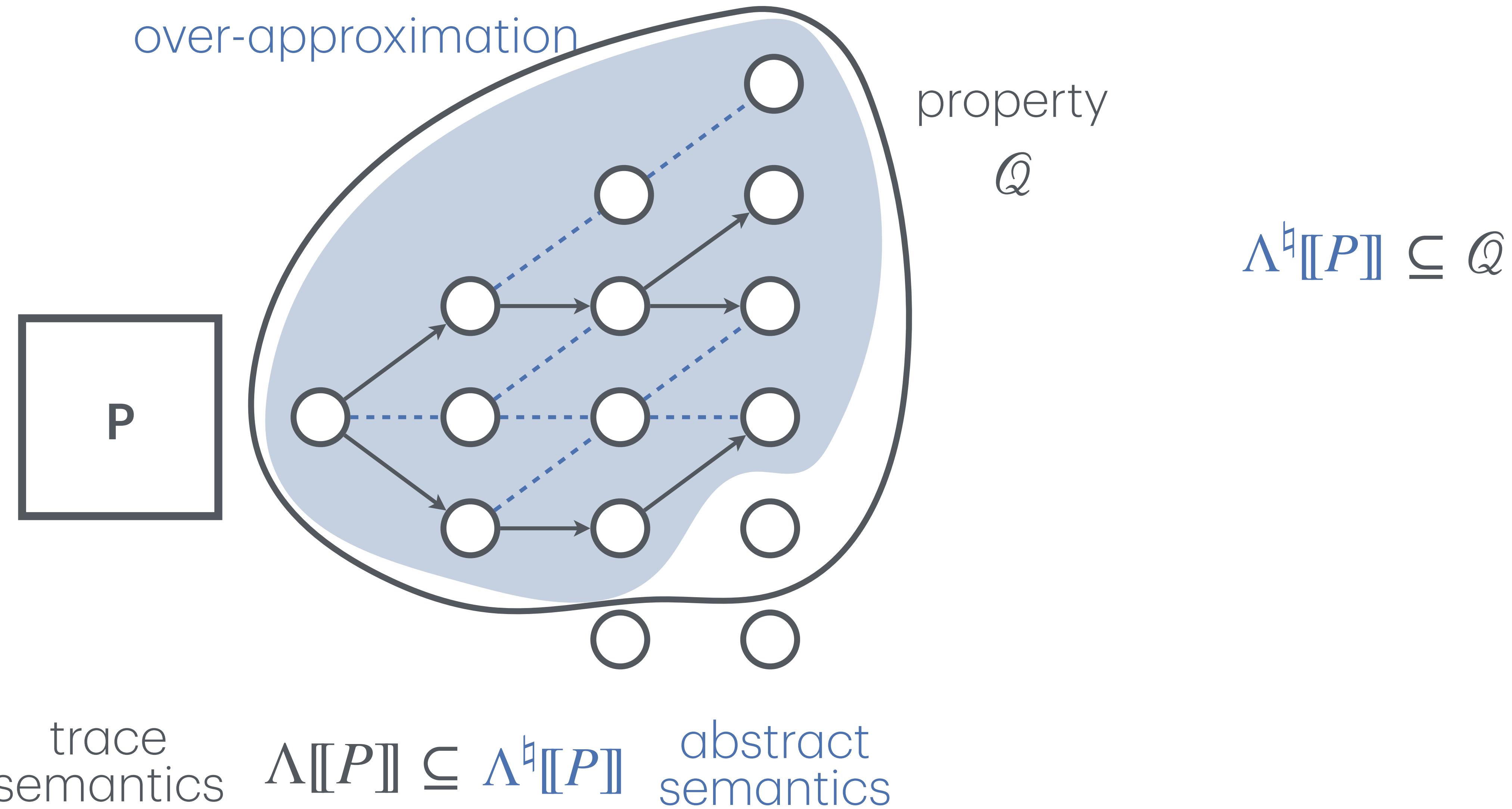


trace  
semantics

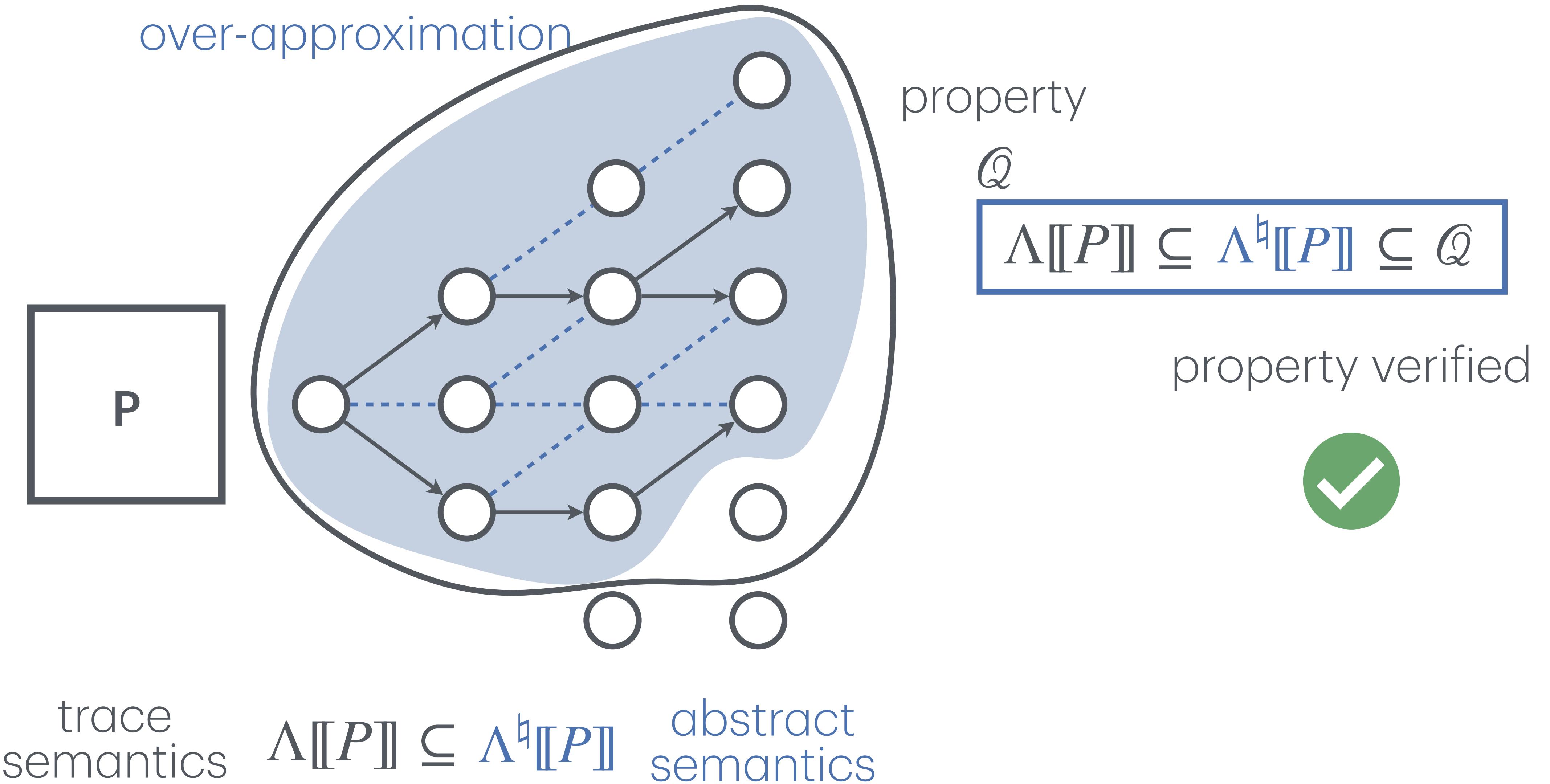
$$\Lambda[P] \subseteq \Lambda^\sharp[P]$$

abstract  
semantics

# Abstract interpretation

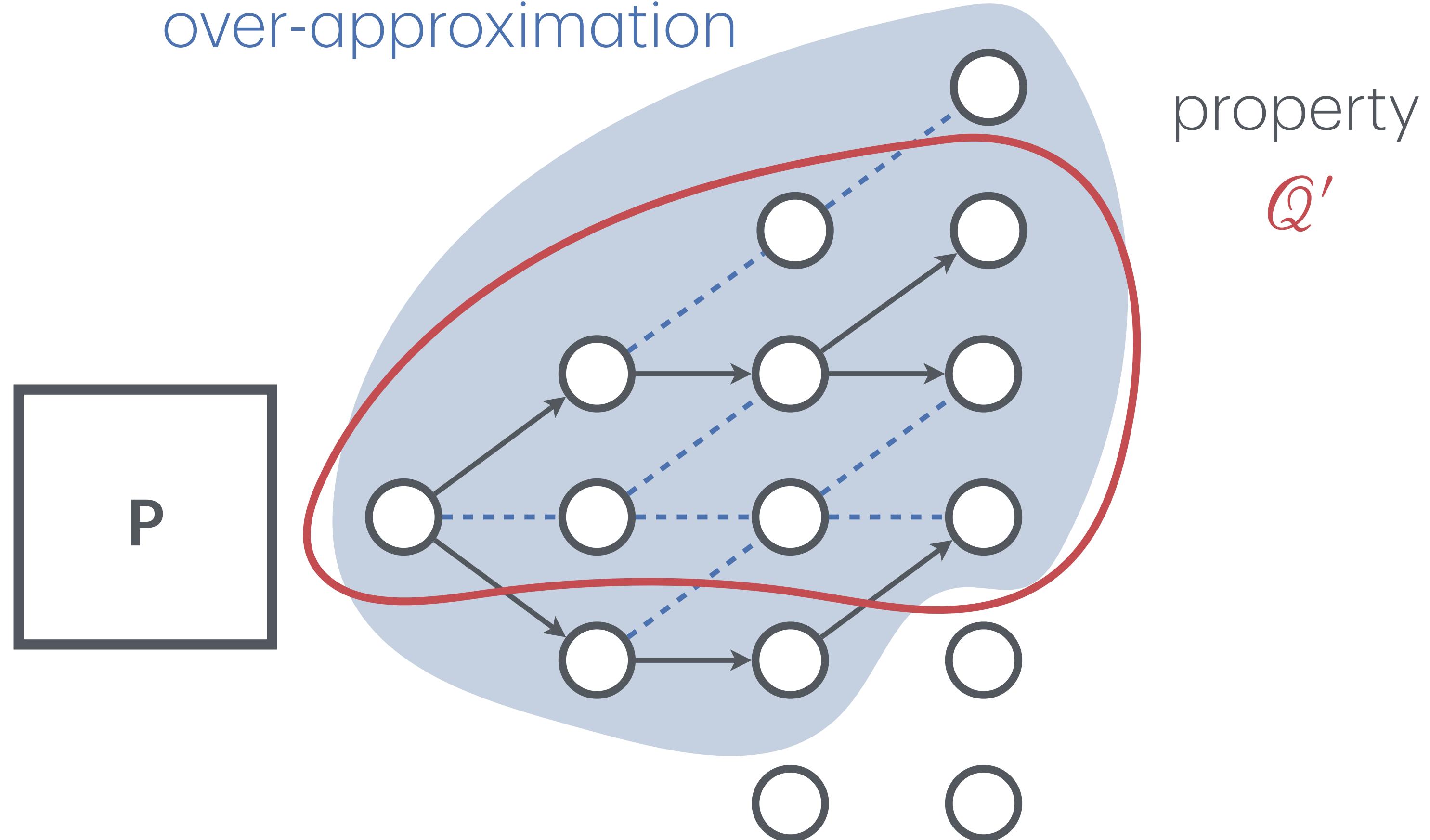


# Abstract interpretation



# Abstract interpretation

over-approximation



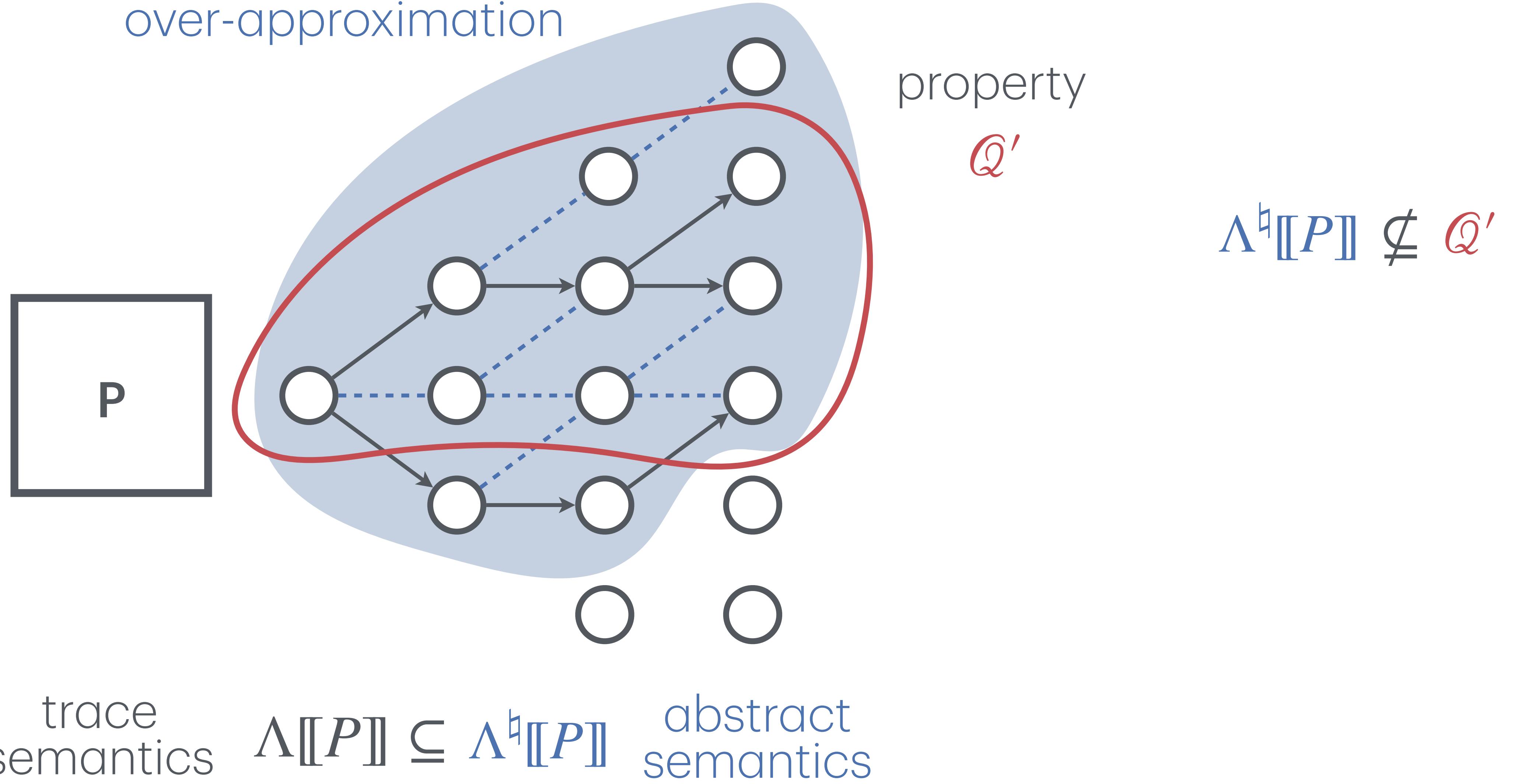
trace  
semantics

$$\Lambda[P] \subseteq \Lambda^\sharp[P]$$

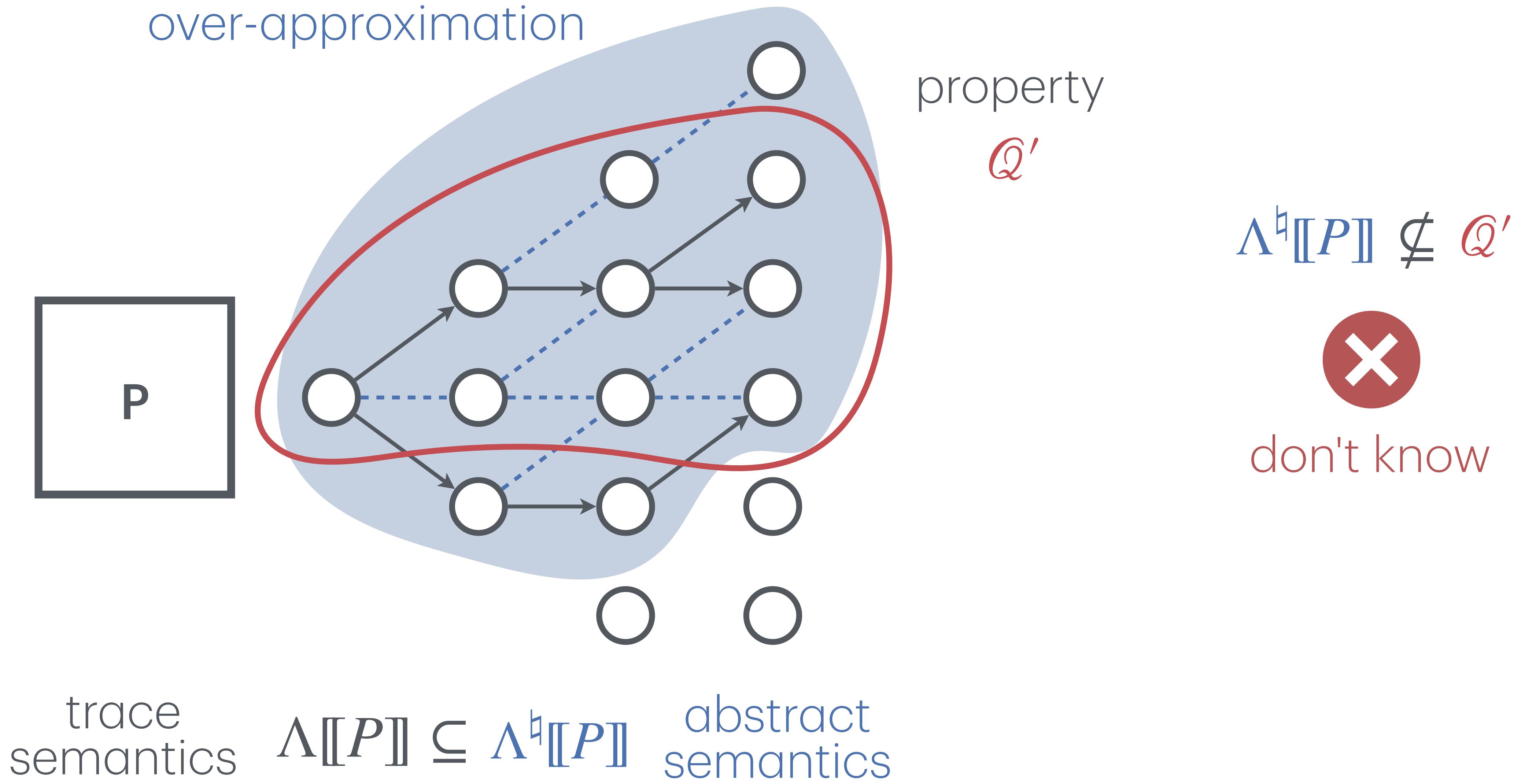
abstract  
semantics

# Abstract interpretation

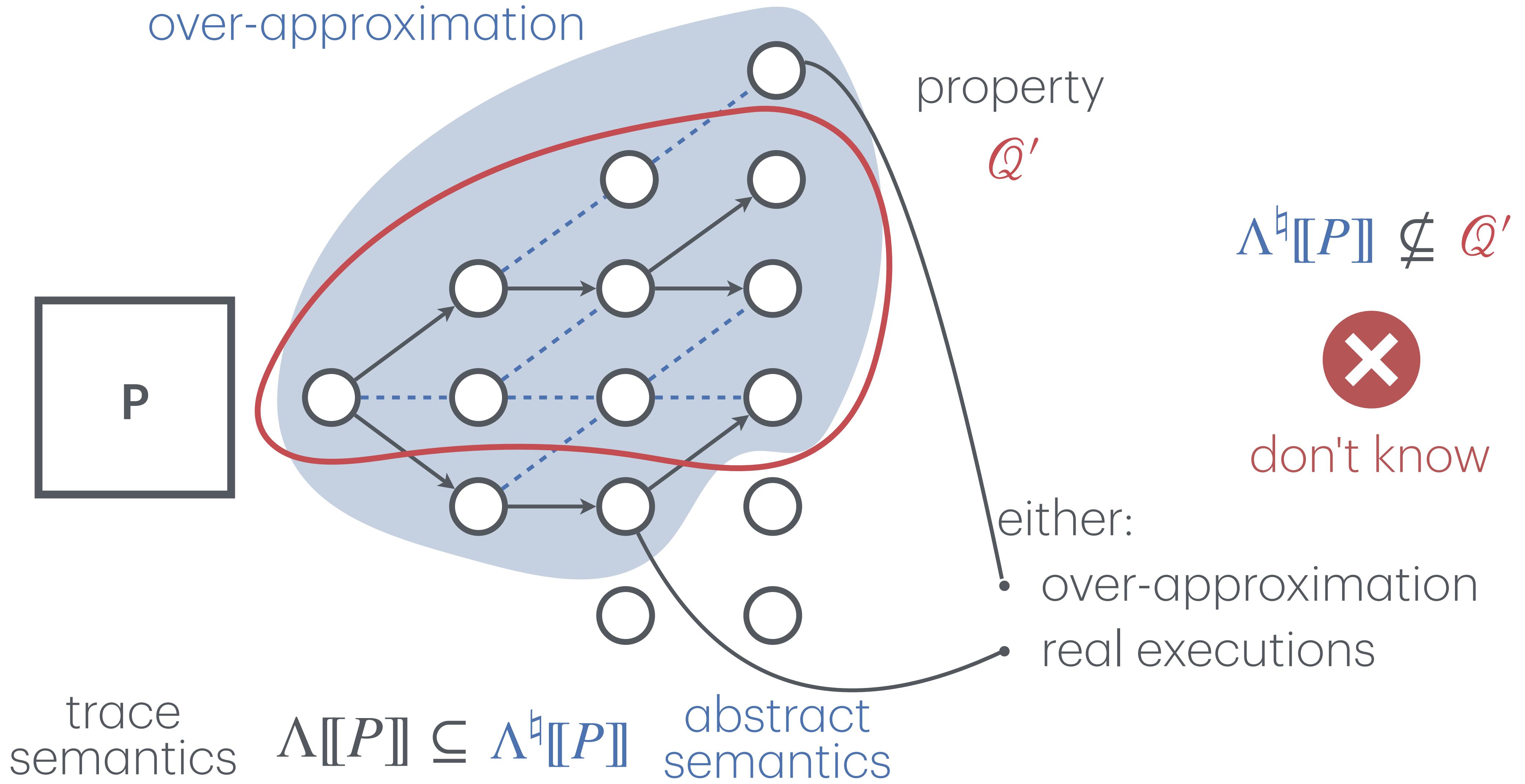
over-approximation



# Abstract interpretation



# Abstract interpretation



# What kind of Properties in this thesis?

---

# What kind of Properties in this thesis?

---

Properties to show the absence of programming errors

# What kind of Properties in this thesis?

---

Properties to show the absence of programming errors



Crashes



Runtime errors

# What kind of Properties in this thesis?

---

Properties to show the absence of programming errors

Crashes

Runtime errors

**Misuse of input data**

# What kind of Properties in this thesis?

---

Crashes

Properties to show the absence of programming errors

Runtime errors

**Misuse of input data**

In this thesis we focus on **Input Data Usage** properties

# What kind of Properties in this thesis?

---

Crashes

Properties to show the absence of programming errors

Runtime errors

**Misuse of input data**

In this thesis we focus on **Input Data Usage** properties

Intuitively:

"The program misuses the input data  
compared to the developer expectations"

# Input Data Usage

---

## Growth in a Time of Debt

By CARMEN M. REINHART AND KENNETH S. ROGOFF\*

In this paper, we exploit a new multi-country historical dataset on public (government) debt to search for a systemic relationship between high

especially against the backdrop of graying populations and rising social insurance costs? Are sharply elevated public debts ultimately a man-

# Input Data Usage

## Growth in a Time of Debt

By CARMEN M. REINHART AND KENNETH S. ROGOFF\*

In this paper, we exploit a new multi-country historical dataset on public (government) debt to search for a systemic relationship between high

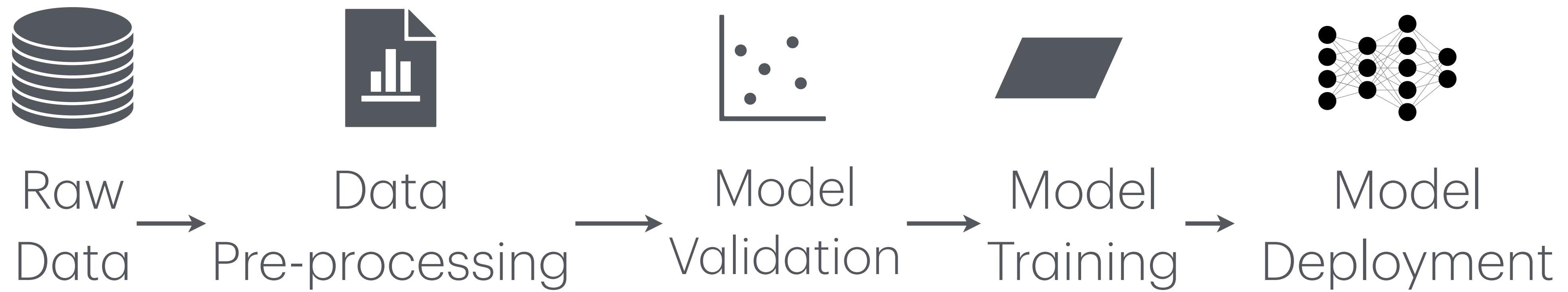
especially against the backdrop of graying populations and rising social insurance costs? Are sharply elevated public debts ultimately a man-

Wrong conclusion due to  
the **misuse** of Denmark's  
only datapoint

A	B	C	D	E	F	G	H	I
1	Country	Year	Debt-to-GDP	Growth rate	Debt category	Herndon, Ash and Pollin result		
2	Australia	1946	190.42	-3.56	4			
3	Australia	1947	177.32	2.46	4			
4	Australia	1948	148.93	6.44	4			
5	Australia	1949	125.83	6.61	4			
6	Australia	1950	109.81	6.92	4			
7	Australia	1951	87.09	4.27	3			
8	Australia	1952	86.07	0.90	3			
9	Australia	1953	79.87	3.12	3			
10	Australia	1954	76.85	6.22	3			
11	Australia	1955	74.98	5.46	3			
12	Australia	1956	71.82	3.44	3			
13	Australia	1957	67.72	2.00	3			
14	Australia	1958	66.05	4.80	3			
15	Australia	1959	62.39	6.16	3			
16	Australia	1960	48.34	4.19	2			
17	Australia	1961	49.38	0.69	2			
18	Australia	1962	50.33	6.25	2			
19	Australia	1963	47.59	6.13	2			
20	Australia	1964	45.01	6.84	2			
21	Australia	1965	43.74	5.11	2			
22	Australia	1966	42.48	2.78	2			
23	Australia	1967	40.53	6.75	2			
24	Australia	1968	39.49	5.88	2			
25	Australia	1969	36.80	6.08	2			
26	Australia	1970	34.96	6.36	2			
27	Australia	1971	32.29	4.45	2			
28	Australia	1972	30.72	2.81	2			
29	Australia	1973	27.36	5.42	1			
30	Australia	1974	22.37	2.48	1			
31	Australia	1975	23.64	2.71	1			
32	Australia	1976	22.98	4.03	1			
33	Australia	1977	23.64	1.05	1			
34	Australia	1978	24.73	2.90	1			
35	Australia	1979	20.87	5.27	1			
36	Australia	1980	19.12	2.00	1			
37	Australia	1981	16.18	4.16	1			
38	Australia	1982	15.04	-0.01	1			
39	Australia	1983	17.58	-0.53	1			
40	Australia	1984	20.06	6.41	1			
41	Australia	1985	21.42	5.72	1			
42	Australia	1986	21.49	2.11	1			
43	Australia	1987	19.99	4.40	1			
44	Australia	1988	16.29	4.03	1			
45	Australia	1989	12.50	4.57	1			
46	Australia	1990	10.33	1.60	1			
Country-year mean growth						4.2	3.1	
Reinhart and Rogoff results								
Public debt						<30%	30-60	
Country-year mean growth								
Denmark						4.2	3.1	
Cross-country mean growth						4.0	3.0	
Adultered						4.1	2.9	

# Machine Learning

---



Pipeline of **data intensive manipulation**

# Static Analysis by Abstract Interpretation for Quantitative Program Properties

# Quantitative

**Qualitative** properties are  
**not always sufficient** for  
capturing the complexity  
of real-world software

# Quantitative Input Data Usage

1	A	B	C	D	E	F	G	H	I
1	Country	Year	Debt-to-GDP	Growth rate	Debt category		Herndon, Ash and Pollin result		
2	Australia	1946	190.42	-3.56	4		Public debt		
3	Australia	1947	177.32	2.46	4		<30%	30-60	
4	Australia	1948	148.93	6.44	4		Country-year mean growth	4.2	3.1
5	Australia	1949	125.83	6.61	4				
6	Australia	1950	109.81	6.92	4				
7	Australia	1951	87.09	4.27	3				
8	Australia	1952	86.07	0.90	3				
9	Australia	1953	79.87	3.12	3				
10	Australia	1954	76.85	6.22	3				
11	Australia	1955	74.98	5.46	3				
12	Australia	1956	71.82	3.44	3				
13	Australia	1957	67.72	2.00	3				
14	Australia	1958	66.05	4.80	3				
15	Australia	1959	62.39	6.16	3				
16	Australia	1960	48.34	4.19	2				
17	Australia	1961	49.38	0.69	2				
18	Australia	1962	50.33	6.25	2				
19	Australia	1963	47.59	6.13	2				
20	Australia	1964	45.01	6.84	2				
21	Australia	1965	43.74	5.11	2				
22	Australia	1966	42.48	2.78	2				
23	Australia	1967	40.53	6.75	2				
24	Australia	1968	39.49	5.88	2				
25	Australia	1969	36.80	6.08	2				
26	Australia	1970	34.96	6.36	2				
27	Australia	1971	32.29	4.45	2				
28	Australia	1972	30.72	2.81	2				
29	Australia	1973	27.36	5.42	1				
30	Australia	1974	22.37	2.48	1				
31	Australia	1975	23.64	2.71	1				
32	Australia	1976	22.98	4.03	1				
33	Australia	1977	23.64	1.05	1				
34	Australia	1978	24.73	2.90	1				
35	Australia	1979	20.87	5.27	1				
36	Australia	1980	19.12	2.00	1				
37	Australia	1981	16.18	4.16	1				
38	Australia	1982	15.04	-0.01	1				
39	Australia	1983	17.58	-0.53	1				
40	Australia	1984	20.06	6.41	1				
41	Australia	1985	21.42	5.72	1				
42	Australia	1986	21.49	2.11	1				
43	Australia	1987	19.99	4.40	1				
44	Australia	1988	16.29	4.03	1				
45	Australia	1989	12.50	4.57	1				
46	Australia	1990	10.33	1.60	1				

**Growth in a Time of Debt**

By CARMEN M. REINHART AND KENNETH S. ROGOFF\*

In this paper, we exploit a new multi-country historical dataset on public (government) debt to search for a systemic relationship between high especially against the backdrop of gray regulations and rising social insurance costs sharply elevated public debts ultimately

Impact of Denmark was too big on the conclusion

# Theoretical Framework

---

## Extensional Properties

---

Neural  
Networks

## Intensional Properties

---

Loop  
Iterations

# How to measure Impact?

---

# How to measure Impact?

---

Program  $P$ ,  $\mathbf{W} \subseteq \Delta$  input variables of **interest**

# How to measure Impact?

---

Program  $P$ ,  $\mathbf{W} \subseteq \Delta$  input variables of **interest**

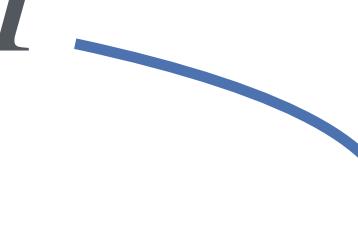
$$\text{IMPACT}_{\mathbf{W}} \in T \rightarrow \mathbb{V}_{\geq 0}^{+\infty}$$

# How to measure Impact?

---

Program  $P, \mathbf{W} \subseteq \Delta$  input variables of **interest**

set of traces  $T$


$$\text{IMPACT}_{\mathbf{W}} \in T \rightarrow \mathbb{V}_{\geq 0}^{+\infty}$$

# How to measure Impact?

---

Program  $P, \mathbf{W} \subseteq \Delta$  input variables of **interest**

$$\text{set of traces } T \xrightarrow{\quad \text{IMPACT}_{\mathbf{W}} \in T \rightarrow \mathbb{V}_{\geq 0}^{+\infty} \quad} \text{positive values}$$

# How to measure Impact?

Program  $P, \mathbf{W} \subseteq \Delta$  input variables of **interest**

$$\text{set of traces } T \xrightarrow{\quad \text{IMPACT}_{\mathbf{W}} \in T \rightarrow \mathbb{V}_{\geq 0}^{+\infty} \quad} \text{positive values}$$

$$\mathcal{B}_{\text{IMPACT}_{\mathbf{W}}}^{\leq k} \triangleq \{\Lambda \in T \mid \text{IMPACT}_{\mathbf{W}}(\Lambda) \leq k\}$$

# How to measure Impact?

Program  $P, W \subseteq \Delta$  input variables of **interest**

$$\text{set of traces } T \xrightarrow{\quad \text{IMPACT}_W \in T \rightarrow \mathbb{V}_{\geq 0}^{+\infty} \quad} \text{positive values}$$

$$\mathcal{B}_{\text{IMPACT}_W}^{\leq k} \triangleq \underbrace{\{\Lambda \in T \mid \text{IMPACT}_W(\Lambda) \leq k\}}$$

all the program semantics

# How to measure Impact?

Program  $P, \mathbf{W} \subseteq \Delta$  input variables of **interest**

$$\text{set of traces } T \xrightarrow{\quad \text{IMPACT}_{\mathbf{W}} \in T \rightarrow \mathbb{V}_{\geq 0}^{+\infty} \quad} \text{positive values}$$

$$\mathcal{B}_{\text{IMPACT}_{\mathbf{W}}}^{\leq k} \triangleq \underbrace{\{\Lambda \in T \mid \text{IMPACT}_{\mathbf{W}}(\Lambda) \leq k\}}$$

all the program semantics where input variables  $\mathbf{W} \models$

# How to measure Impact?

Program  $P, \mathbf{W} \subseteq \Delta$  input variables of **interest**

$$\text{set of traces } T \xrightarrow{\quad \text{IMPACT}_{\mathbf{W}} \in T \rightarrow \mathbb{V}_{\geq 0}^{+\infty} \quad} \text{positive values}$$

$$\mathcal{B}_{\text{IMPACT}_{\mathbf{W}}}^{\leq k} \triangleq \underbrace{\{\Lambda \in T \mid \text{IMPACT}_{\mathbf{W}}(\Lambda) \leq k\}}$$

all the program semantics where input variables  $\mathbf{W}$  have an impact below  $k$

# The $k$ -Bounded Impact Property

---

$$\mathcal{B}_{\text{IMPACT}_W}^{\leq k} \triangleq \{\Lambda \in T \mid \text{IMPACT}_W(\Lambda) \leq k\}$$

# The $k$ -Bounded Impact Property

---

$$\mathcal{B}_{\text{IMPACT}_W}^{\leq k} \triangleq \{\Lambda \in T \mid \text{IMPACT}_W(\Lambda) \leq k\}$$

goal:  $P \models \mathcal{B}_{\text{IMPACT}_W}^{\leq k}$

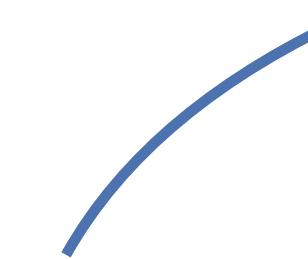
# The $k$ -Bounded Impact Property

---

$$\mathcal{B}_{\text{IMPACT}_W}^{\leq k} \triangleq \{\Lambda \in T \mid \text{IMPACT}_W(\Lambda) \leq k\}$$

goal:  $P \models \mathcal{B}_{\text{IMPACT}_W}^{\leq k} \iff$

$$\Lambda[P] \in \mathcal{B}_{\text{IMPACT}_W}^{\leq k}$$



trace semantics of program  $P$

# The $k$ -Bounded Impact Property

$$\mathcal{B}_{\text{IMPACT}_W}^{\leq k} \triangleq \{\Lambda \in T \mid \text{IMPACT}_W(\Lambda) \leq k\}$$

goal:  $P \models \mathcal{B}_{\text{IMPACT}_W}^{\leq k} \iff$

$$\Lambda[P] \in \mathcal{B}_{\text{IMPACT}_W}^{\leq k}$$

trace semantics of program  $P$

$$\Lambda^C[P] \subseteq \mathcal{B}_{\text{IMPACT}_W}^{\leq k}$$

collecting semantics

$$\Lambda^C[P] \triangleq \{\Lambda[P]\}$$

# The RANGE Quantifier

---

# The RANGE Quantifier

---

Program  $P, \mathbf{W} \subseteq \Delta$  input variables of **interest**

# The RANGE Quantifier

---

Program  $P, \mathbf{W} \subseteq \Delta$  input variables of **interest**

1. Fix values of all input variables but  $\mathbf{W}$

# The RANGE Quantifier

---

Program  $P, \mathbf{W} \subseteq \Delta$  input variables of **interest**

1. Fix values of all input variables **but  $\mathbf{W}$**
2. Monitor how the **outcome** of the program  $P$  behaves with respect to the **value changes** of  $\mathbf{W}$

# The RANGE Quantifier

---

Program  $P, \mathbf{W} \subseteq \Delta$  input variables of **interest**

1. Fix values of all input variables **but  $\mathbf{W}$**
2. Monitor how the **outcome** of the program  $P$  behaves with respect to the **value changes** of  $\mathbf{W}$
3. Worst-case scenario: the **biggest distance** among these outcomes

# The RANGE Quantifier

---

Program  $P, W \subseteq \Delta$  input variables of **interest**

1. Fix values of all input variables **but  $W$**
2. Monitor how the **outcome** of the program  $P$  behaves with respect to the **value changes** of  $W$
3. Worst-case scenario: the **biggest distance** among these outcomes

# The RANGE Quantifier

Program  $P, W \subseteq \Delta$  input variables of **interest**

1. Fix values of all input variables **but  $W$**
2. Monitor how the **outcome** of the program  $P$  behaves with respect to the **value changes** of  $W$
3. Worst-case scenario: the **biggest distance** among these outcomes

- Absolute values (in the thesis)

# The RANGE Quantifier

Program  $P, W \subseteq \Delta$  input variables of **interest**

- Absolute values  
(in the thesis)
- Derivatives

1. Fix values of all input variables **but  $W$**
2. Monitor how the **outcome** of the program  $P$  behaves with respect to the **value changes** of  $W$
3. Worst-case scenario: the **biggest distance** among these outcomes

# The RANGE Quantifier

Program  $P, W \subseteq \Delta$  input variables of **interest**

1. Fix values of all input variables **but  $W$**
2. Monitor how the **outcome** of the program  $P$  behaves with respect to the **value changes** of  $W$
3. Worst-case scenario: the **biggest distance** among these outcomes

- Absolute values (in the thesis)
- Derivatives
- Entropies

# Reinhart & Rogoff

---

```
1: def mean_growth_rate_60_90(  
2:     portugal1, portugal2, portugal3,  
3:     norway1,  
4:     uk1, uk2, uk3, uk4,  
5:     usa1, usa2, usa3):  
6:     portugal_avg = avg(portugal1, portugal2, portugal3)  
7:     norway_avg = avg(norway1)  
8:     uk_avg = avg(uk1, uk2, uk3, uk4)  
9:     usa_avg = avg(usa1, usa2, usa3)  
10:    return avg(portugal_avg, norway_avg, uk_avg, usa_avg)
```

# Reinhart & Rogoff

---

```
1: def mean_growth_rate 60 90(             
2:     portugal1, portugal2, portugal3,      
3:     norway1,                            
4:     uk1, uk2, uk3, uk4,                  
5:     usa1, usa2, usa3):                 
6:     portugal_avg = avg(portugal1, portugal2, portugal3)   
7:     norway_avg = avg(norway1)            
8:     uk_avg = avg(uk1, uk2, uk3, uk4)     
9:     usa_avg = avg(usa1, usa2, usa3)        
10:    return avg(portugal_avg, norway_avg, uk_avg, usa_avg)
```

# Reinhart & Rogoff

---

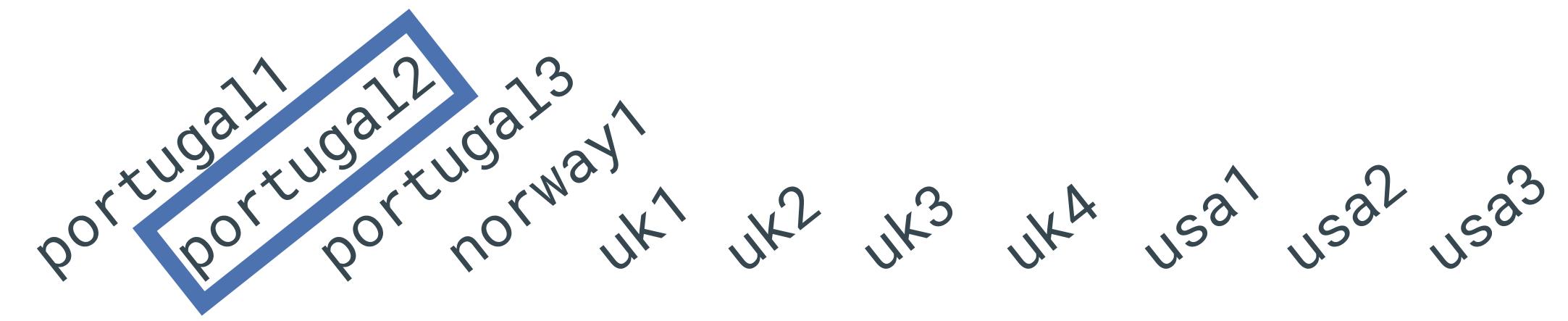
```
1: def mean_growth_rate 60 90(           
2:     portugal1, portugal2, portugal3,      
3:     norway1,                            
4:     uk1, uk2, uk3, uk4,                 
5:     usa1, usa2, usa3):                  
6:     portugal_avg = avg(portugal1, portugal2, portugal3) 
7:     norway_avg = avg(norway1)           
8:     uk_avg = avg(uk1, uk2, uk3, uk4)   
9:     usa_avg = avg(usa1, usa2, usa3)    
10:    return avg(portugal_avg, norway_avg, uk_avg, usa_avg)
```

# The RANGE Quantifier

---

$W = \text{portugal2}$

`mean_growth_rate_60_90`



# The RANGE Quantifier

---

$W = \text{portugal2}$



`mean_growth_rate_60_90( 1.2, 0.9, 1.3, 7.1, 1.2, 1.2, 1.2, 1.7, 1.1, 0.9, 2.0, 2.1 ) = 2.8`

# The RANGE Quantifier

$W = \text{portugal2}$



# The RANGE Quantifier

$W = \text{portugal2}$



# The RANGE Quantifier

W = portugal2

tugal2

mean\_growth\_rate\_60\_90( 1.2 0.9 1.3, 7.1, 1.2, 1.2, 1.7, 1.1, 0.9, 2.0, 2.1 ) = 2.8

1.2 0.5 1.3, 7.1, 1.2, 1.2, 1.7, 1.1, 0.9, 2.0, 2.1 | 2.77 } 0.06

1.2 1.3 1.3, 7.1, 1.2, 1.2, 1.7, 1.1, 0.9, 2.0, 2.1 | 2.83 }

distance

# The RANGE Quantifier

$W = \text{portugal2}$

			portugal11	portugal12	portugal13	norway1	uk1	uk2	uk3	uk4	usa1	usa2	usa3	distance
mean_growth_rate_60_90(	1.2	0.9	1.3, 7.1, 1.2, 1.2, 1.2, 1.7, 1.1, 0.9, 2.0, 2.1											= 2.8
	1.2	0.5	1.3, 7.1, 1.2, 1.2, 1.2, 1.7, 1.1, 0.9, 2.0, 2.1											2.77
	1.2	1.3	1.3, 7.1, 1.2, 1.2, 1.2, 1.7, 1.1, 0.9, 2.0, 2.1											2.83
	1.2	5.5	1.3, 7.1, 1.2, 1.2, 1.2, 1.7, 1.1, 0.9, 2.0, 2.1											3.18

# The RANGE Quantifier

w = portugal2

# The RANGE Quantifier

w = portugal2

tugal2

mean\_growth\_rate\_60\_90( 1.2 0.9 1.3, 7.1, 1.2, 1.2, 1.7, 1.1, 0.9, 2.0, 2.1 ) = 2.8

1.2 0.5 1.3, 7.1, 1.2, 1.2, 1.7, 1.1, 0.9, 2.0, 2.1

1.2 1.3 1.3, 7.1, 1.2, 1.2, 1.7, 1.1, 0.9, 2.0, 2.1

1.2 5.5 1.3, 7.1, 1.2, 1.2, 1.7, 1.1, 0.9, 2.0, 2.1

1.2 ...

1.2 0.0 1.3, 7.1, 1.2, 1.2, 1.7, 1.1, 0.9, 2.0, 2.1

distance

portugal11  
portugal12  
portugal13  
norway1  
uk1  
uk2  
uk3  
uk4  
usa1  
usa2  
usa3

-20 <  $x$  < 20

# The RANGE Quantifier

w = portugal2

Infinitely many!

# The RANGE Quantifier

w = portugal2

tugal2

mean\_growth\_rate\_60\_90( 1.2 0.9 1.3, 7.1, 1.2, 1.2, 1.7, 1.1, 0.9, 2.0, 2.1 ) = 2.8

1.2 0.5 1.3, 7.1, 1.2, 1.2, 1.7, 1.1, 0.9, 2.0, 2.1

1.2 1.3 1.3, 7.1, 1.2, 1.2, 1.7, 1.1, 0.9, 2.0, 2.1

1.2 5.5 1.3, 7.1, 1.2, 1.2, 1.7, 1.1, 0.9, 2.0, 2.1

1.2 ...

1.2 0.0 1.3, 7.1, 1.2, 1.2, 1.7, 1.1, 0.9, 2.0, 2.1

portugal11  
portugal12  
portugal13  
norway1  
uk1  
uk2  
uk3  
uk4  
usa1  
usa2  
usa3

distance

-20 ≤  $x$  ≤ 20

⇒  $a$

2.8  
2.77  
2.83  
3.18  
0.55  
2.73

# Infinitely many!

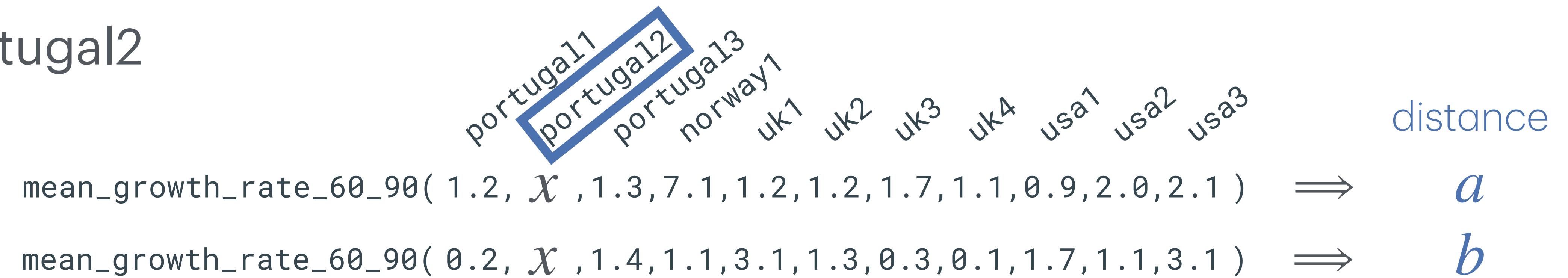
# The RANGE Quantifier

$W = \text{portugal2}$



# The RANGE Quantifier

$W = \text{portugal2}$



# The RANGE Quantifier

$W = \text{portugal2}$

		distance
$\text{mean\_growth\_rate\_60\_90}( 1.2, \mathcal{X}, 1.3, 7.1, 1.2, 1.2, 1.2, 1.7, 1.1, 0.9, 2.0, 2.1 )$	$\Rightarrow$	$a$
$\text{mean\_growth\_rate\_60\_90}( 0.2, \mathcal{X}, 1.4, 1.1, 3.1, 1.3, 0.3, 0.1, 1.7, 1.1, 3.1 )$	$\Rightarrow$	$b$
$\text{mean\_growth\_rate\_60\_90}( 1.2, \mathcal{X}, 1.4, 5.0, 1.4, 2.0, 1.1, 0.9, 0.1, 2.8, 1.2 )$	$\Rightarrow$	$c$

# The RANGE Quantifier

$W = \text{portugal2}$

	portugal11 portugal12 portugal13 norway1 uk1 uk2 uk3 uk4 usa1 usa2 usa3	distance
$\text{mean\_growth\_rate\_60\_90}( 1.2, \mathcal{X}, 1.3, 7.1, 1.2, 1.2, 1.2, 1.7, 1.1, 0.9, 2.0, 2.1 ) \Rightarrow a$		$a$
$\text{mean\_growth\_rate\_60\_90}( 0.2, \mathcal{X}, 1.4, 1.1, 3.1, 1.3, 0.3, 0.1, 1.7, 1.1, 3.1 ) \Rightarrow b$		$b$
$\text{mean\_growth\_rate\_60\_90}( 1.2, \mathcal{X}, 1.4, 5.0, 1.4, 2.0, 1.1, 0.9, 0.1, 2.8, 1.2 ) \Rightarrow c$		$c$
for all possible input values	...	...
$\text{mean\_growth\_rate\_60\_90}( 4.2, \mathcal{X}, 5.3, 1.1, 3.2, 8.1, 0.7, 1.9, 4.9, 3.1, 7.1 ) \Rightarrow z$		$z$

# The RANGE Quantifier

$W = \text{portugal2}$

	portugal11 portugal12 portugal13 norway1 uk1 uk2 uk3 uk4 usa1 usa2 usa3	distance
$\text{mean\_growth\_rate\_60\_90}( 1.2, \mathcal{X}, 1.3, 7.1, 1.2, 1.2, 1.2, 1.7, 1.1, 0.9, 2.0, 2.1 ) \Rightarrow a$		$a$
$\text{mean\_growth\_rate\_60\_90}( 0.2, \mathcal{X}, 1.4, 1.1, 3.1, 1.3, 0.3, 0.1, 1.7, 1.1, 3.1 ) \Rightarrow b$		$b$
$\text{mean\_growth\_rate\_60\_90}( 1.2, \mathcal{X}, 1.4, 5.0, 1.4, 2.0, 1.1, 0.9, 0.1, 2.8, 1.2 ) \Rightarrow c$		$c$
for all possible input values	...	...
$\text{mean\_growth\_rate\_60\_90}( 4.2, \mathcal{X}, 5.3, 1.1, 3.2, 8.1, 0.7, 1.9, 4.9, 3.1, 7.1 ) \Rightarrow z$		$z$
	$-20 \leq x \leq 20$	

# The RANGE Quantifier

$W = \text{portugal2}$

	portugal11 portugal12 portugal13 norway1 uk1 uk2 uk3 uk4 usa1 usa2 usa3	distance
$\text{mean\_growth\_rate\_60\_90}( 1.2, \mathcal{X}, 1.3, 7.1, 1.2, 1.2, 1.2, 1.7, 1.1, 0.9, 2.0, 2.1 ) \Rightarrow a$		$a$
$\text{mean\_growth\_rate\_60\_90}( 0.2, \mathcal{X}, 1.4, 1.1, 3.1, 1.3, 0.3, 0.1, 1.7, 1.1, 3.1 ) \Rightarrow b$		$b$
$\text{mean\_growth\_rate\_60\_90}( 1.2, \mathcal{X}, 1.4, 5.0, 1.4, 2.0, 1.1, 0.9, 0.1, 2.8, 1.2 ) \Rightarrow c$		$c$
for all possible input values	...	...
$\text{mean\_growth\_rate\_60\_90}( 4.2, \mathcal{X}, 5.3, 1.1, 3.2, 8.1, 0.7, 1.9, 4.9, 3.1, 7.1 ) \Rightarrow z$		$z$
$-20 \leq x \leq 20$		worst-case scenario = maximum

# The RANGE Quantifier

$W = \text{portugal2}$

	portugal11 portugal12 portugal13 norway1 uk1 uk2 uk3 uk4 usa1 usa2 usa3	distance
$\text{mean\_growth\_rate\_60\_90}( 1.2, \mathcal{X}, 1.3, 7.1, 1.2, 1.2, 1.2, 1.7, 1.1, 0.9, 2.0, 2.1 ) \Rightarrow a$		$a$
$\text{mean\_growth\_rate\_60\_90}( 0.2, \mathcal{X}, 1.4, 1.1, 3.1, 1.3, 0.3, 0.1, 1.7, 1.1, 3.1 ) \Rightarrow b$		$b$
$\text{mean\_growth\_rate\_60\_90}( 1.2, \mathcal{X}, 1.4, 5.0, 1.4, 2.0, 1.1, 0.9, 0.1, 2.8, 1.2 ) \Rightarrow c$		$c$
for all possible input values	...	...
$\text{mean\_growth\_rate\_60\_90}( 4.2, \mathcal{X}, 5.3, 1.1, 3.2, 8.1, 0.7, 1.9, 4.9, 3.1, 7.1 ) \Rightarrow z$		$z$
$-20 \leq x \leq 20$		worst-case scenario = maximum

$\text{RANGE}_{\text{portugal2}}(\text{mean\_growth\_rate\_60\_90})$

# The RANGE Quantifier

$W = \text{portugal2}$

	portugal11 portugal12 portugal13 norway1 uk1 uk2 uk3 uk4 usa1 usa2 usa3	distance
$\text{mean\_growth\_rate\_60\_90}( 1.2, \mathcal{X}, 1.3, 7.1, 1.2, 1.2, 1.2, 1.7, 1.1, 0.9, 2.0, 2.1 ) \Rightarrow a$		$a$
$\text{mean\_growth\_rate\_60\_90}( 0.2, \mathcal{X}, 1.4, 1.1, 3.1, 1.3, 0.3, 0.1, 1.7, 1.1, 3.1 ) \Rightarrow b$		$b$
$\text{mean\_growth\_rate\_60\_90}( 1.2, \mathcal{X}, 1.4, 5.0, 1.4, 2.0, 1.1, 0.9, 0.1, 2.8, 1.2 ) \Rightarrow c$		$c$
for all possible input values	...	...
$\text{mean\_growth\_rate\_60\_90}( 4.2, \mathcal{X}, 5.3, 1.1, 3.2, 8.1, 0.7, 1.9, 4.9, 3.1, 7.1 ) \Rightarrow z$		$z$
$-20 \leq x \leq 20$		worst-case scenario = maximum

$$\text{RANGE}_{\text{portugal2}}(\text{mean\_growth\_rate\_60\_90}) = 3.33$$

# The RANGE Quantifier

$W = \text{norway1}$

	portugal11 portugal12 portugal13 norway1	uk1 uk2 uk3 uk4	usa1 usa2 usa3	distance
$\text{mean\_growth\_rate\_60\_90}( 1.2, 0.9, 1.3, \mathcal{X}, 1.2, 1.2, 1.2, 1.7, 1.1, 0.9, 2.0, 2.1 ) \Rightarrow a$				$a$
$\text{mean\_growth\_rate\_60\_90}( 0.2, 0.9, 0.9, \mathcal{X}, 3.1, 1.3, 0.3, 0.1, 1.7, 1.1, 3.1 ) \Rightarrow b$				$b$
$\text{mean\_growth\_rate\_60\_90}( 1.2, 0.9, 2.4, \mathcal{X}, 1.4, 2.0, 1.1, 0.9, 0.1, 2.8, 1.2 ) \Rightarrow c$				$c$
	...	...	...	...
$\text{mean\_growth\_rate\_60\_90}( 4.2, 0.9, 5.3, \mathcal{X}, 3.2, 8.1, 0.7, 1.9, 4.9, 3.1, 7.1 ) \Rightarrow z$				$z$
		$-20 \leq x \leq 20$		

# The RANGE Quantifier

$W = \text{norway1}$

	portugal11 portugal12 portugal13 norway1	uk1 uk2 uk3 uk4	usa1 usa2 usa3	distance
$\text{mean\_growth\_rate\_60\_90}( 1.2, 0.9, 1.3, \mathcal{X}, 1.2, 1.2, 1.2, 1.7, 1.1, 0.9, 2.0, 2.1 ) \Rightarrow a$				$a$
$\text{mean\_growth\_rate\_60\_90}( 0.2, 0.9, 0.9, \mathcal{X}, 3.1, 1.3, 0.3, 0.1, 1.7, 1.1, 3.1 ) \Rightarrow b$				$b$
$\text{mean\_growth\_rate\_60\_90}( 1.2, 0.9, 2.4, \mathcal{X}, 1.4, 2.0, 1.1, 0.9, 0.1, 2.8, 1.2 ) \Rightarrow c$				$c$
	...	...	...	...
$\text{mean\_growth\_rate\_60\_90}( 4.2, 0.9, 5.3, \mathcal{X}, 3.2, 8.1, 0.7, 1.9, 4.9, 3.1, 7.1 ) \Rightarrow z$				$z$
		$-20 \leq x \leq 20$		

$$\text{RANGE}_{\text{norway1}}(\text{mean\_growth\_rate\_60\_90}) = 10.0$$

# How to compute a sound quantity?

---

# How to compute a sound quantity?

---

goal:  $P \models \mathcal{B}_{\text{RANGE}_W}^{\leq k}$

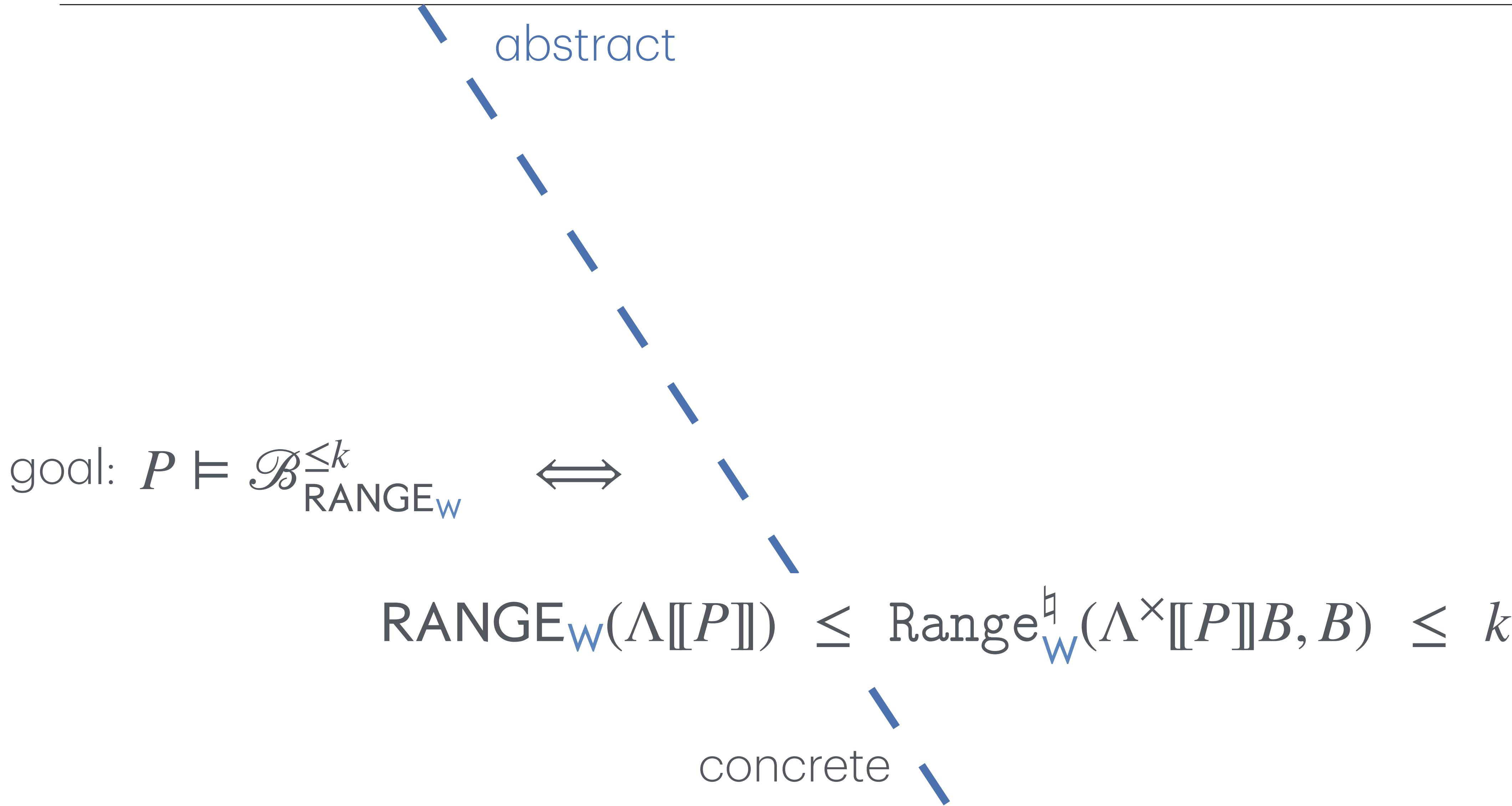
# How to compute a sound quantity?

---

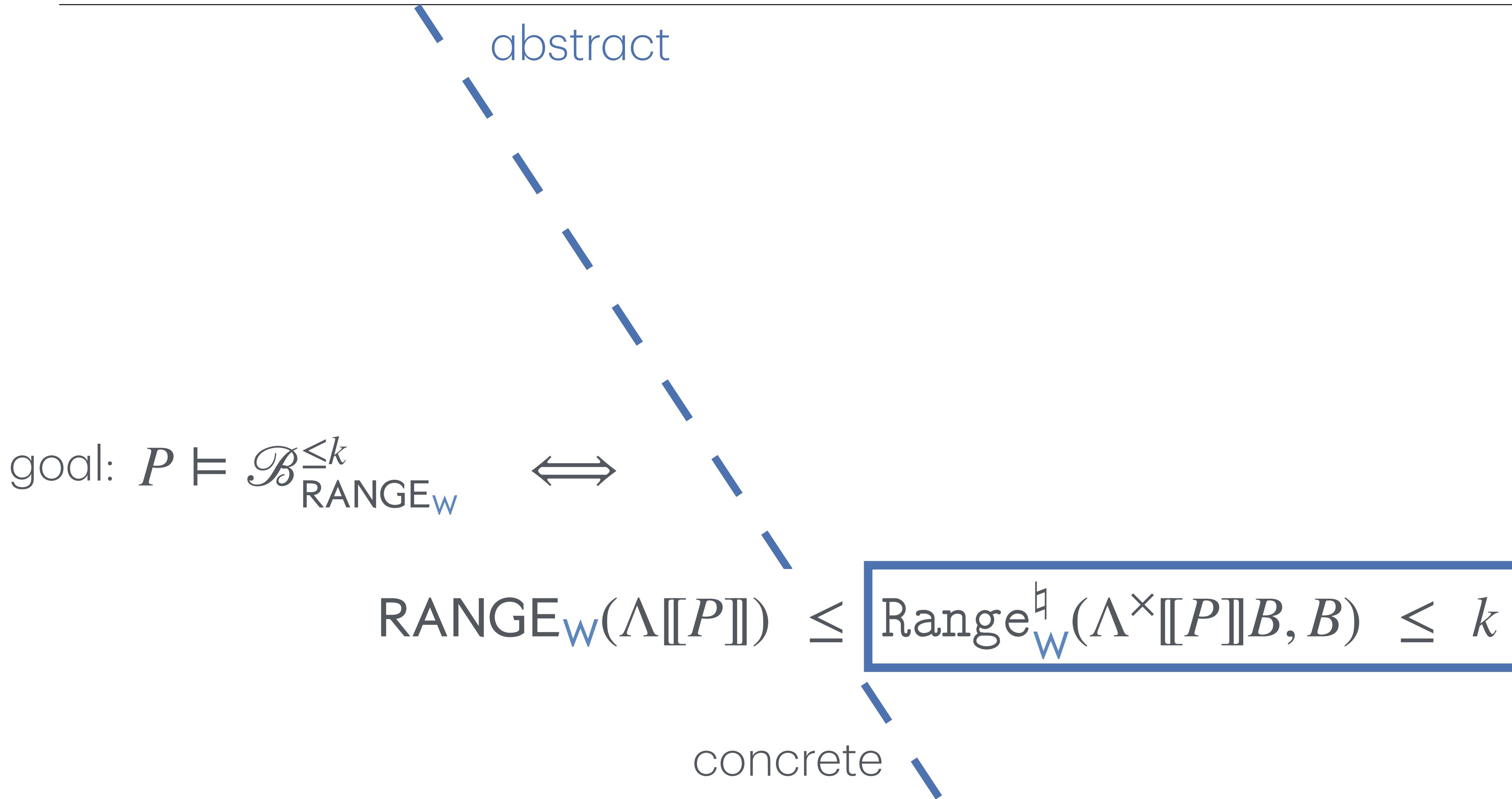
goal:  $P \models \mathcal{B}_{\text{RANGE}_W}^{\leq k} \iff$

$$\text{RANGE}_W(\Lambda[P]) \leq k$$

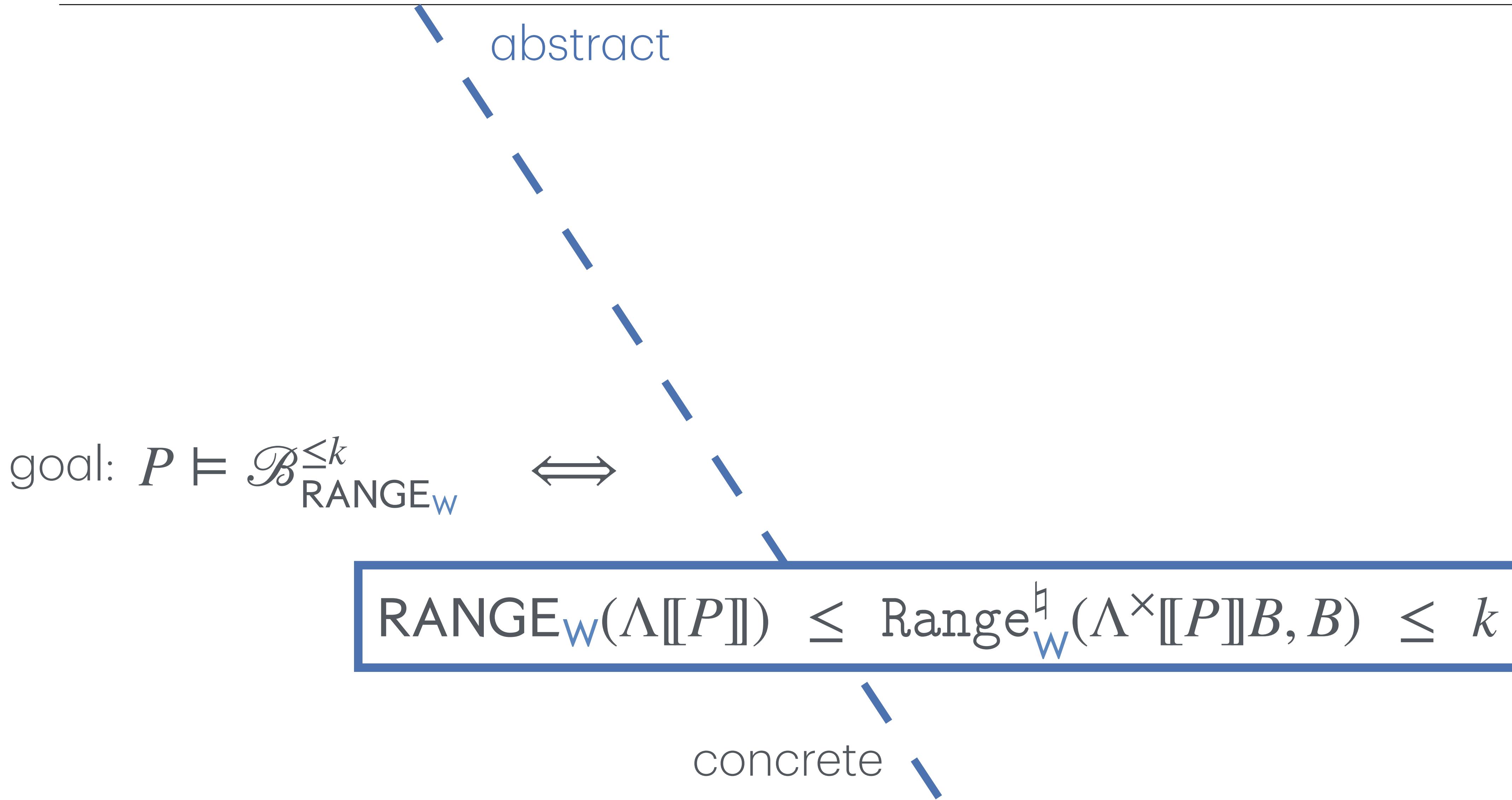
# How to compute a sound quantity?



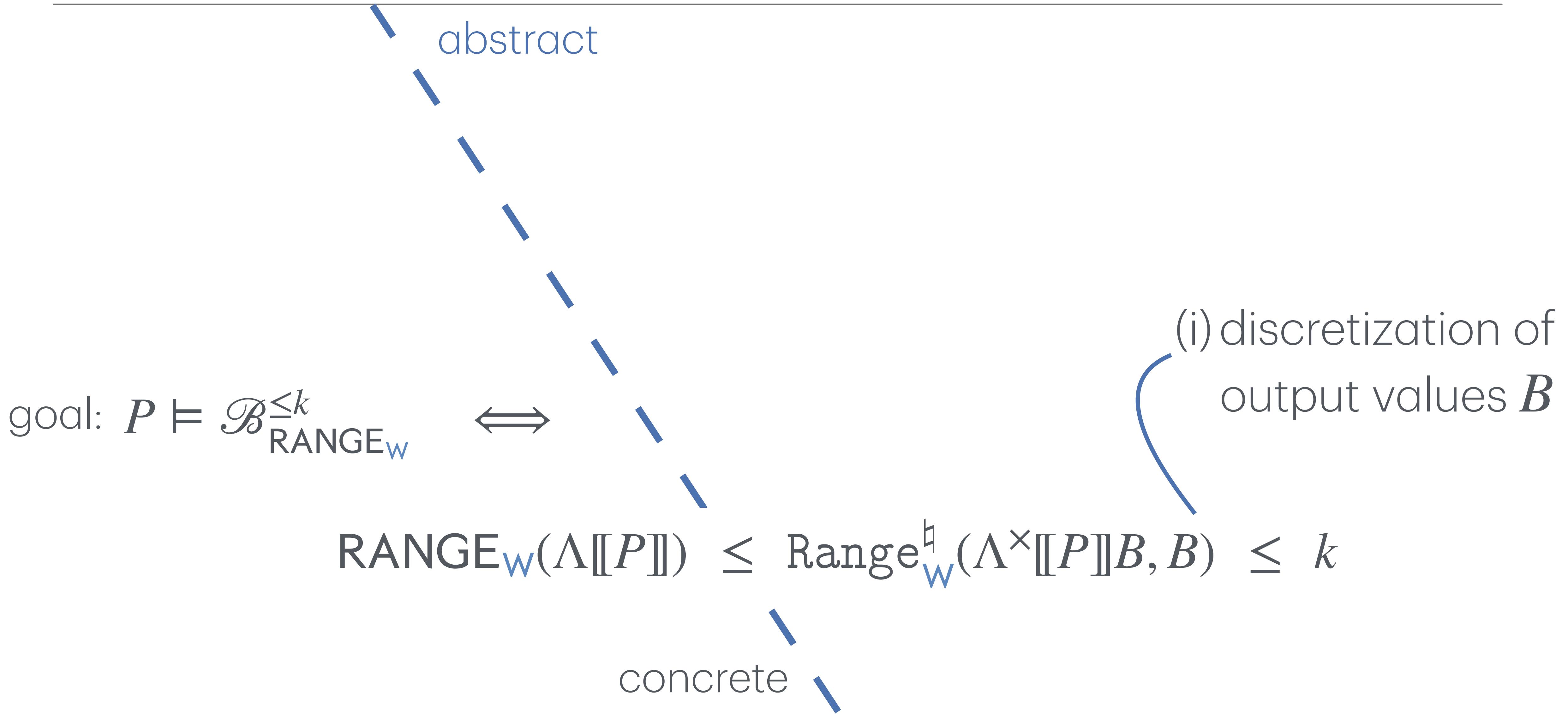
# How to compute a sound quantity?



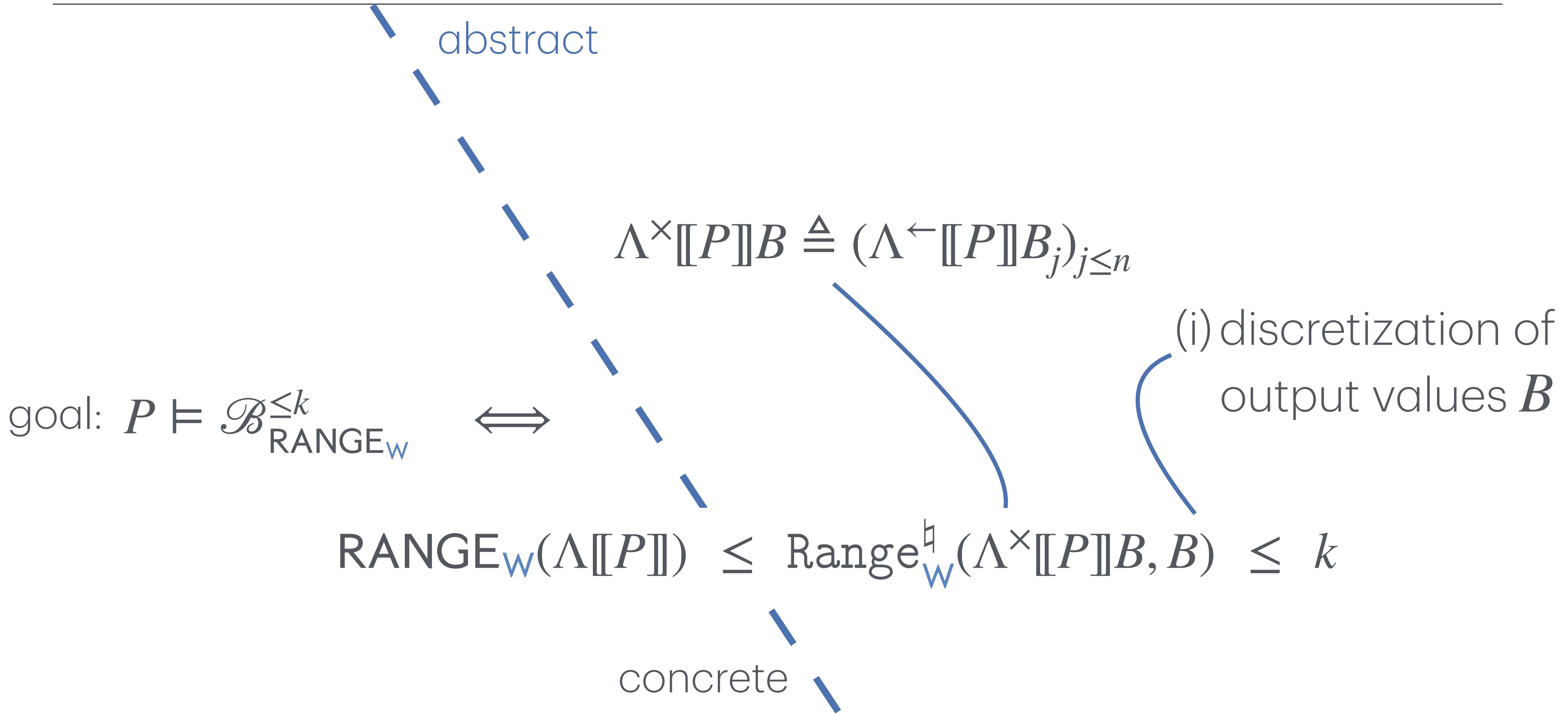
# How to compute a sound quantity?



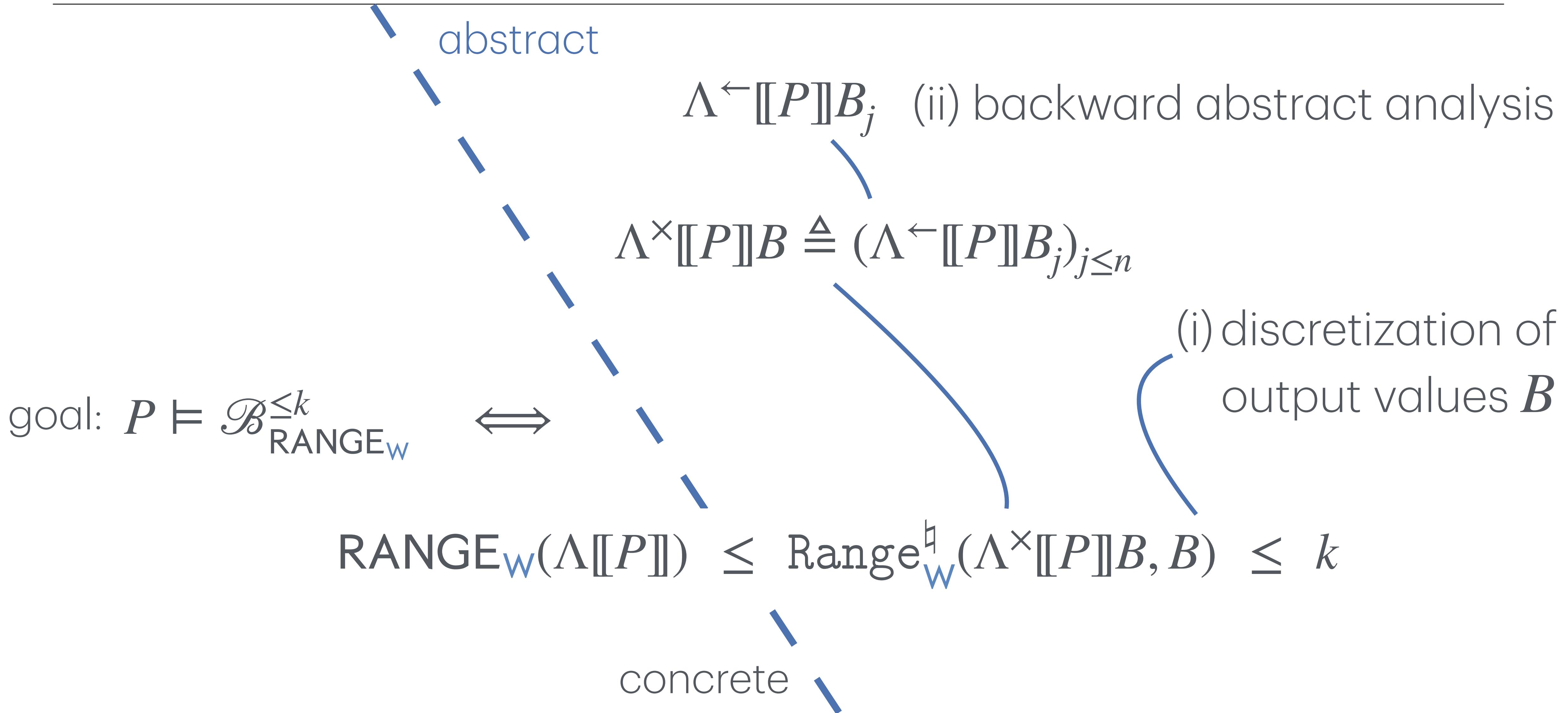
# How to compute a sound quantity?



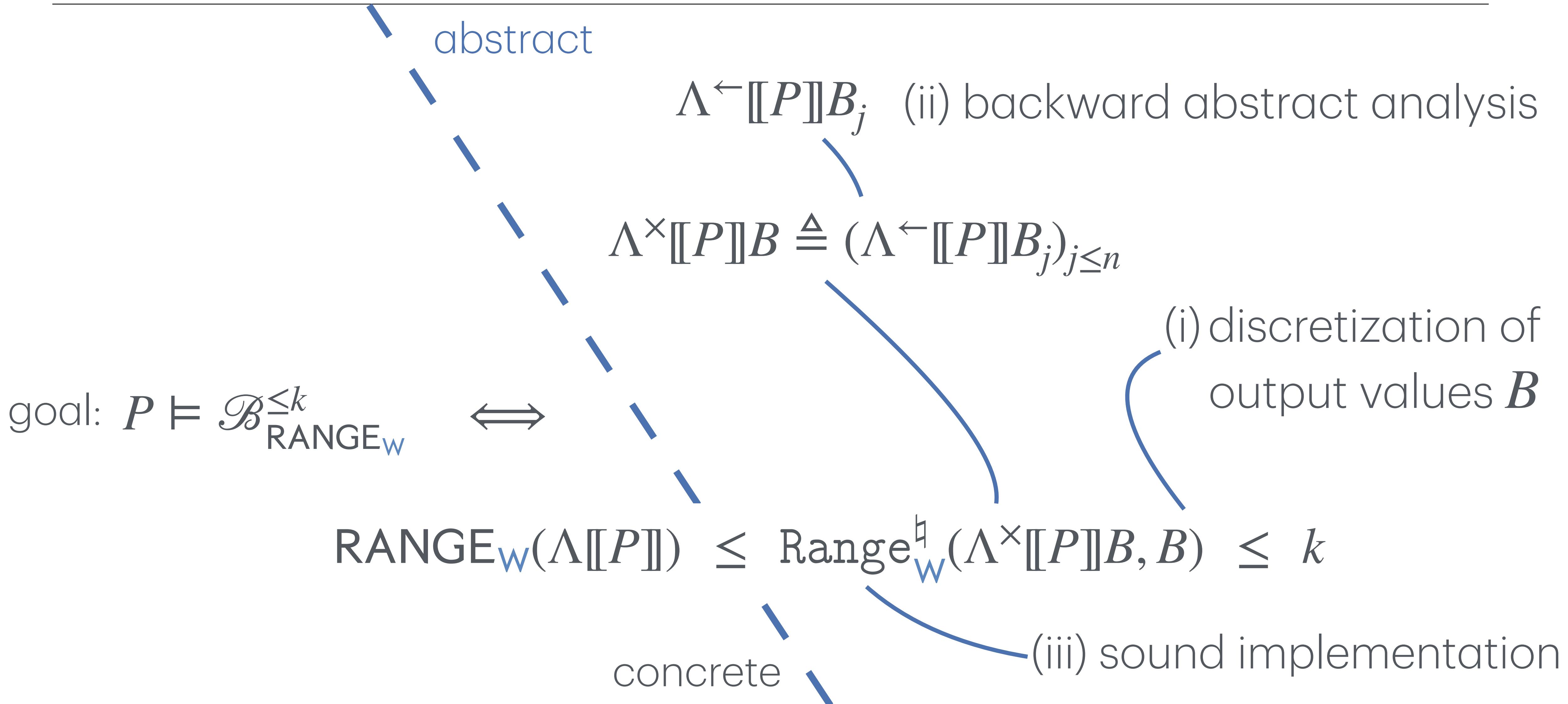
# How to compute a sound quantity?



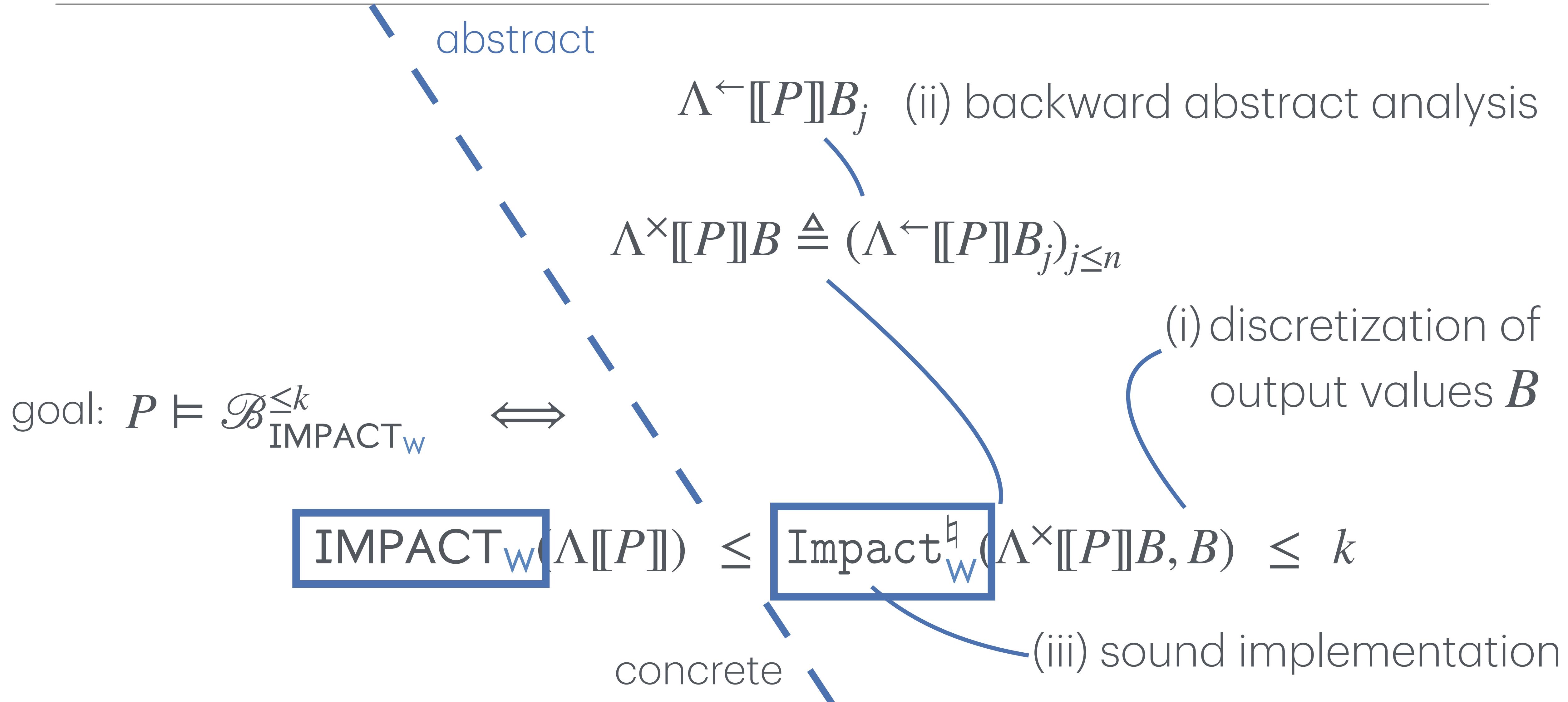
# How to compute a sound quantity?



# How to compute a sound quantity?



# Theoretical Framework



# Reinhart & Rogoff

---

Input Space

portugal11  
portugal12  
portugal13  
norway1  
uk1  
usa1 uk3 uk2  
usa2 uk4  
usa3

# Reinhart & Rogoff



# (i) Output Discretization



# (i) Output Discretization



# (i) Output Discretization

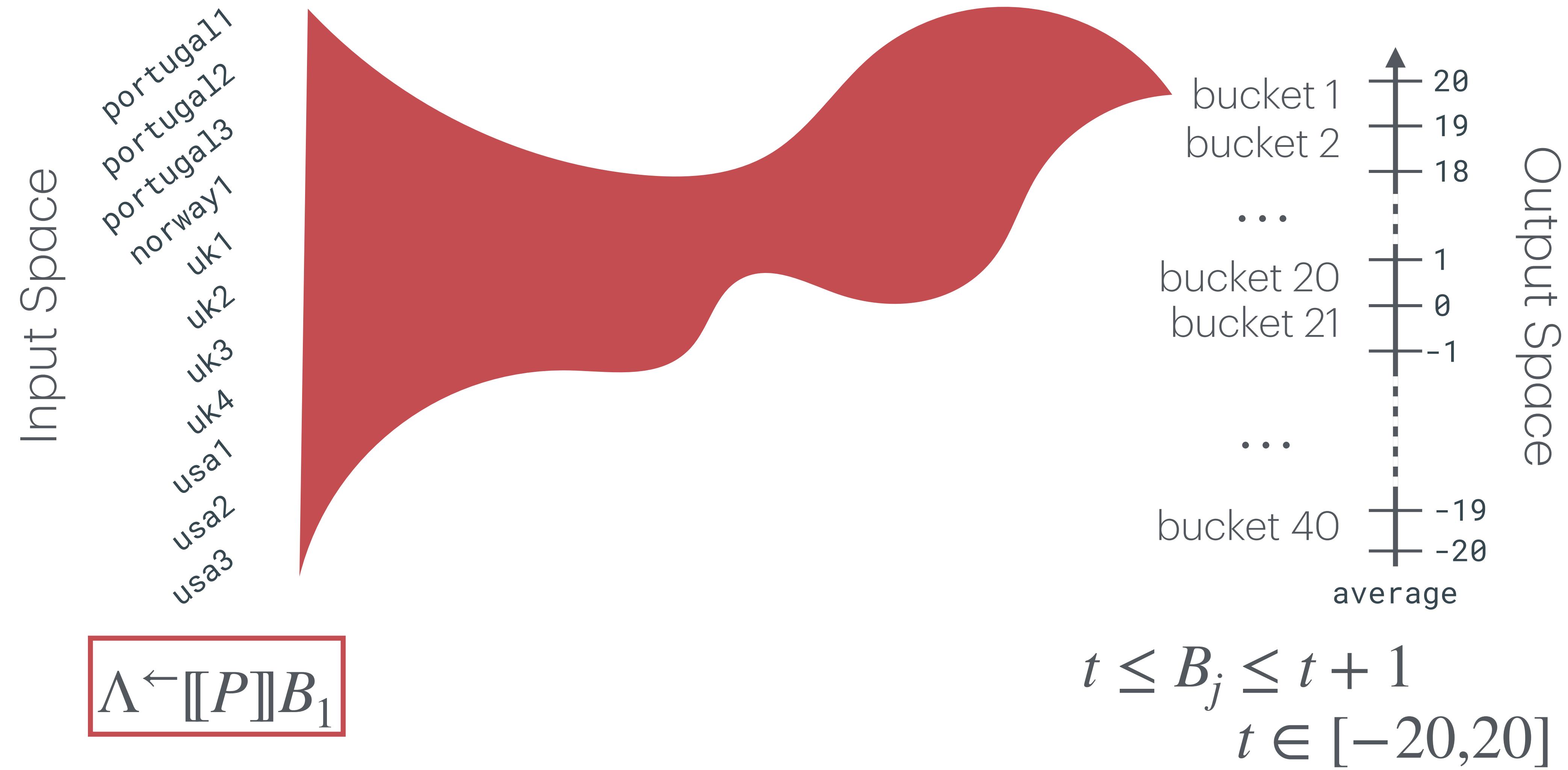


# (ii) Backward Abstract Analysis



$$t \leq B_j \leq t + 1$$
$$t \in [-20, 20]$$

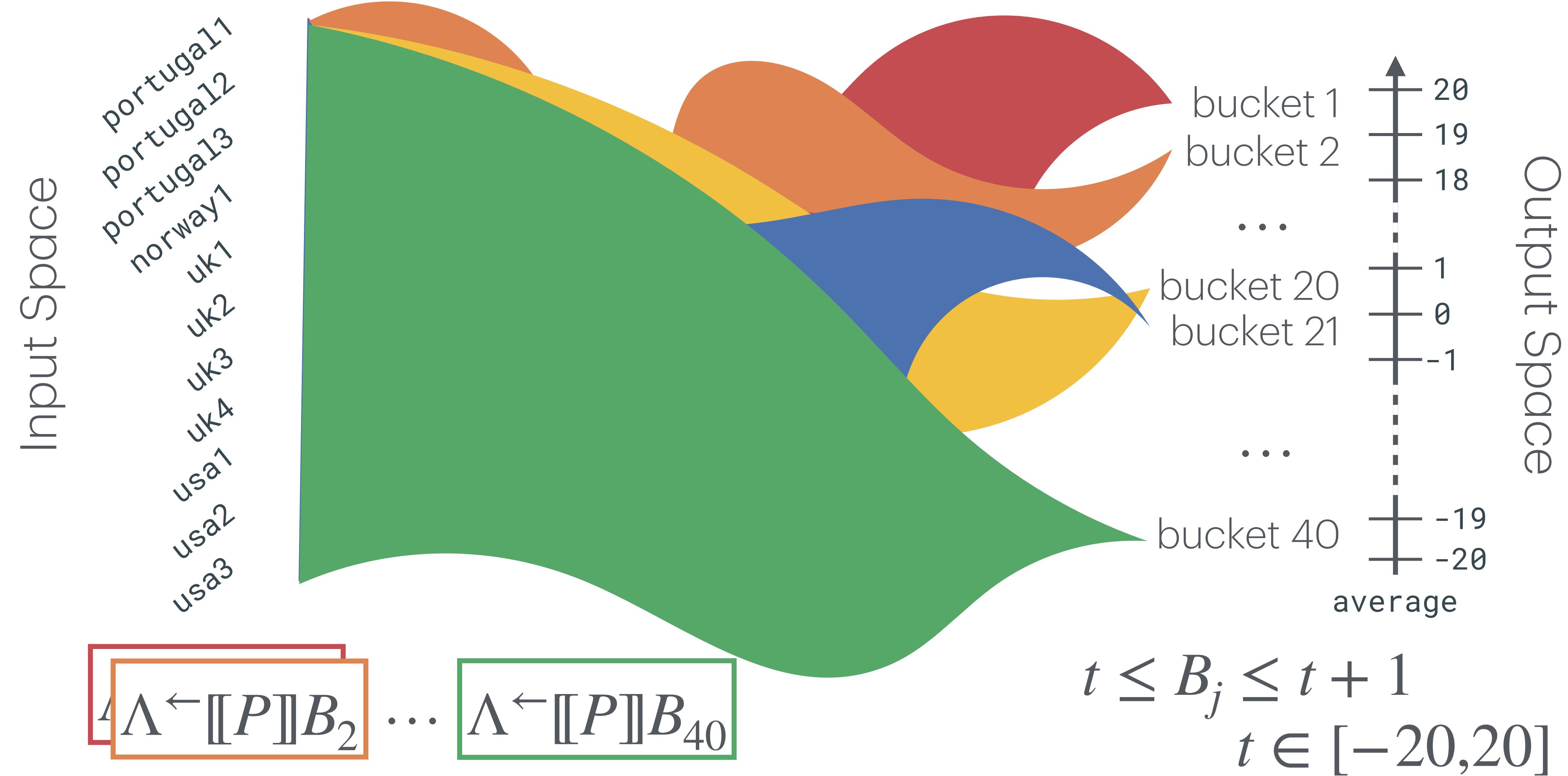
## (ii) Backward Abstract Analysis



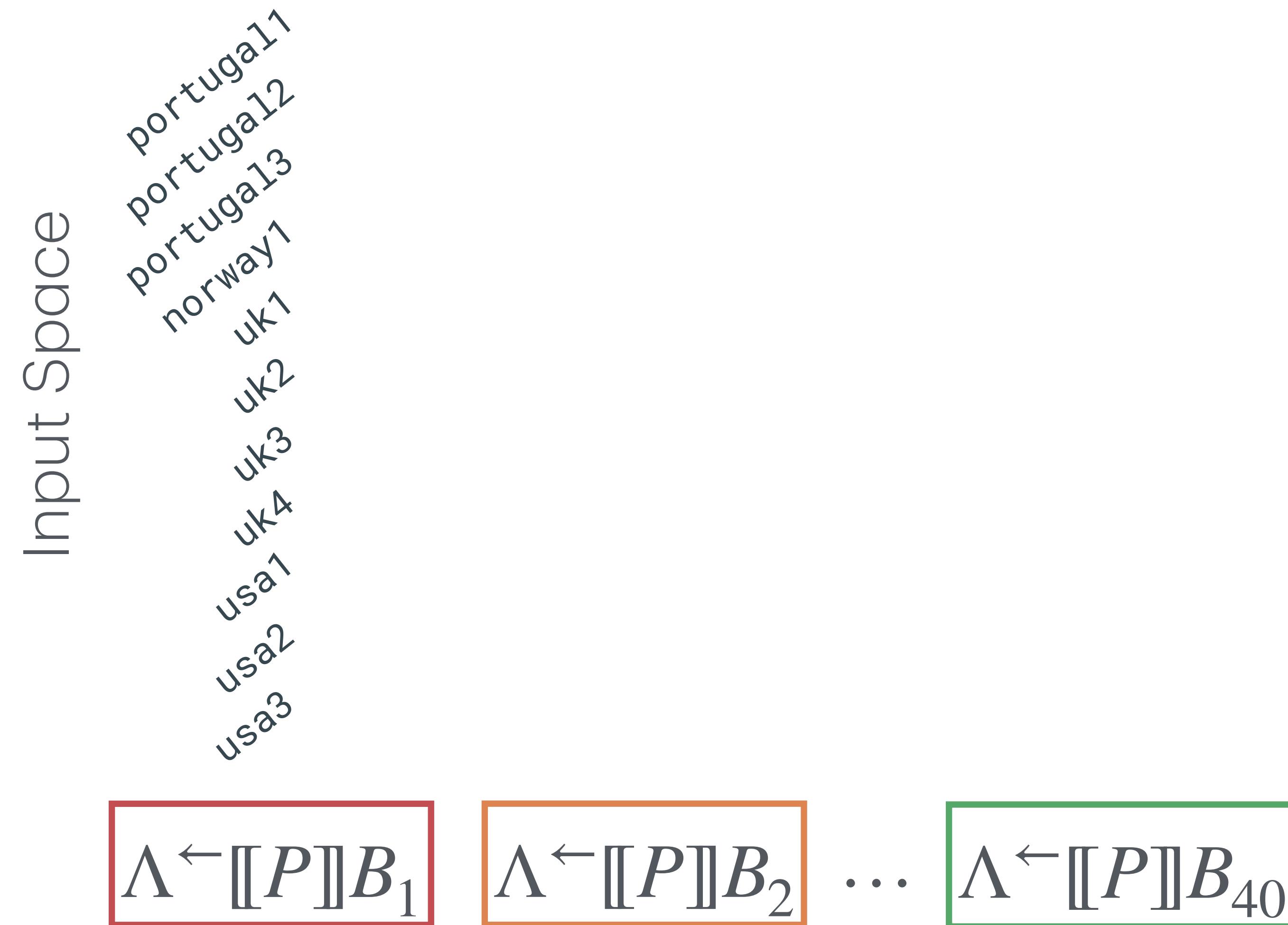
## (ii) Backward Abstract Analysis



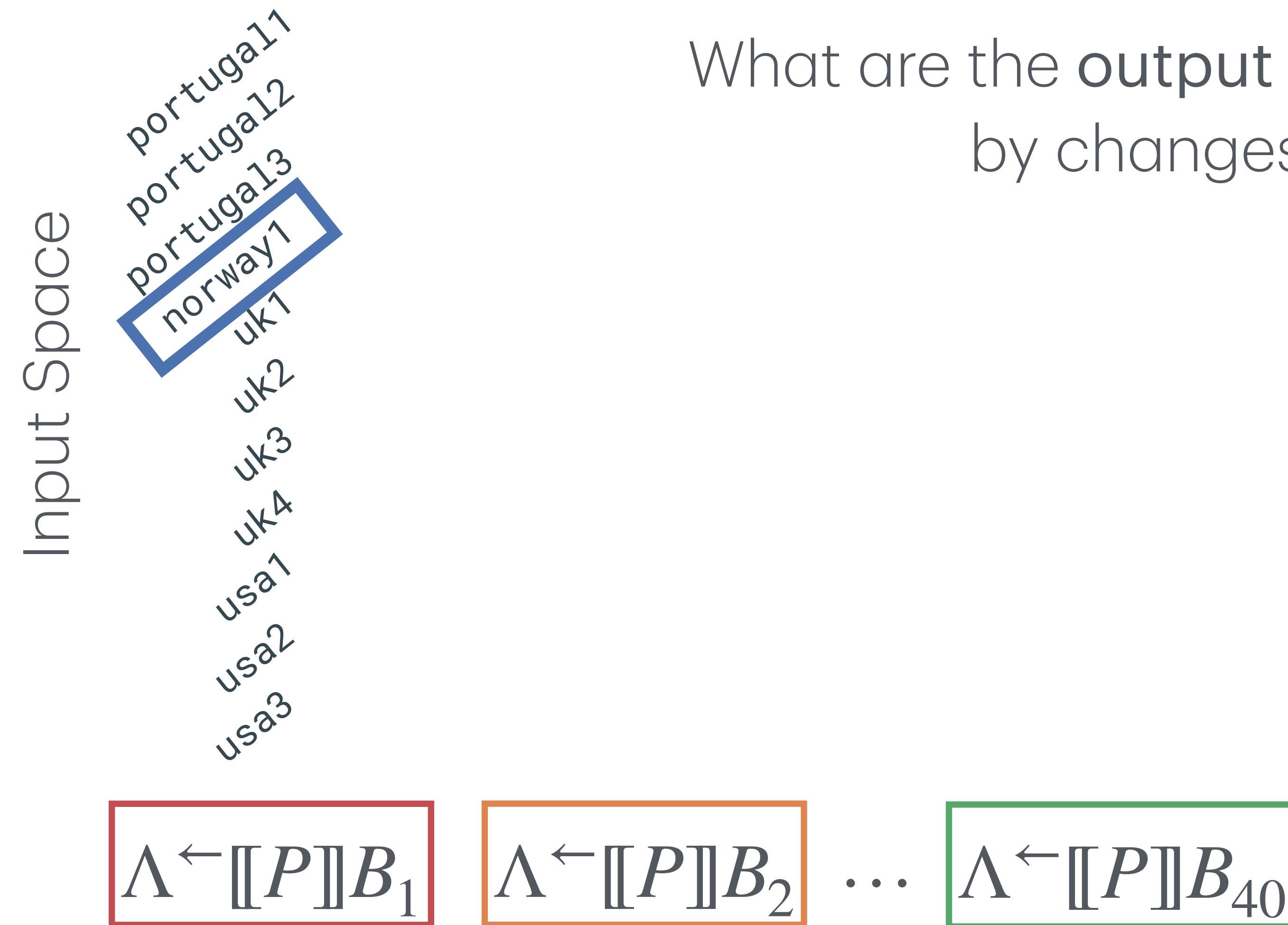
## (ii) Backward Abstract Analysis



# (iii) Range $^\natural$ Abstract Implementation



# (iii) Range $^\natural$ Abstract Implementation



What are the **output values** that can be computed by changes in **norway1** alone?

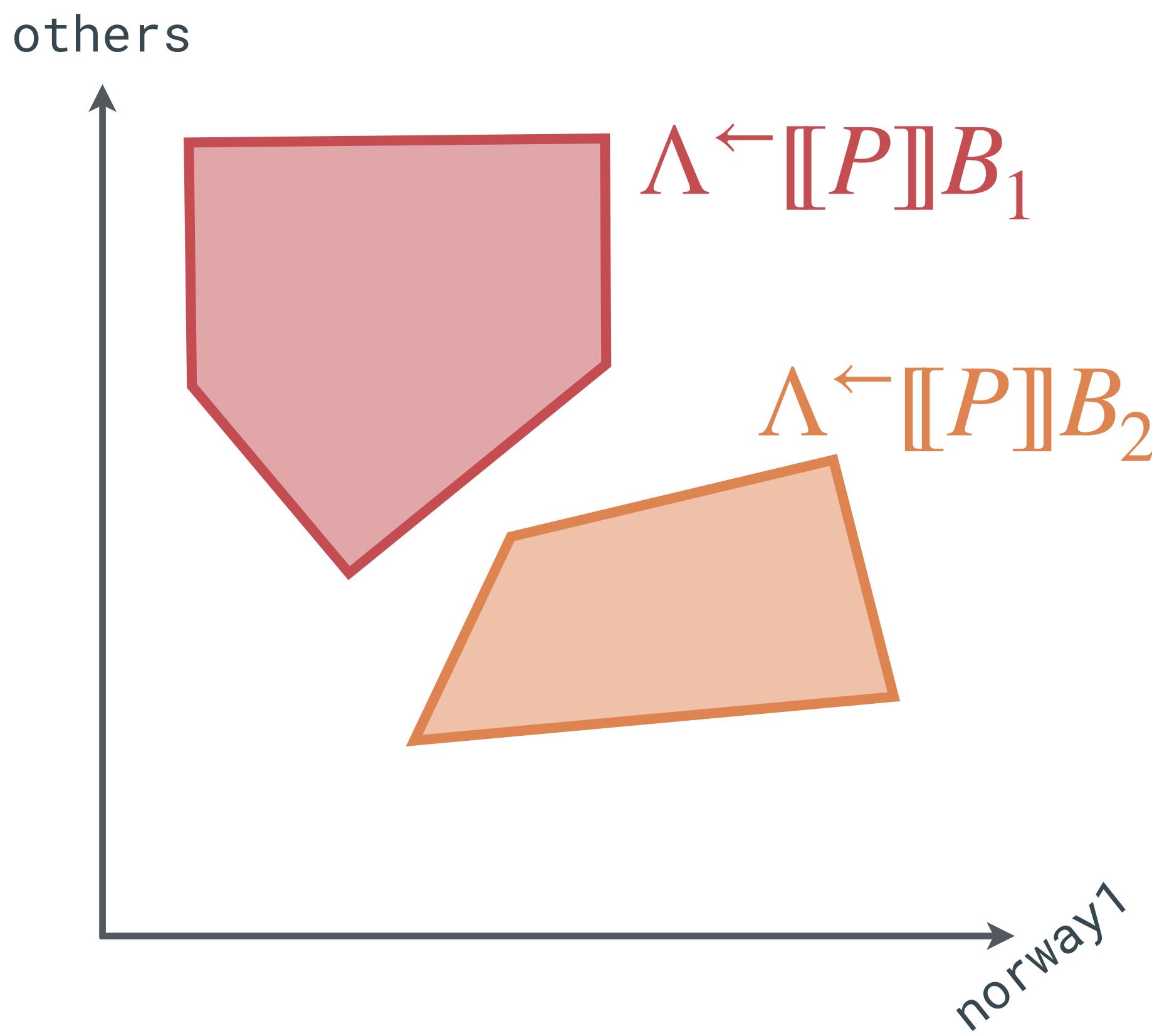
# (iii) Range<sup>♯</sup> Abstract Implementation



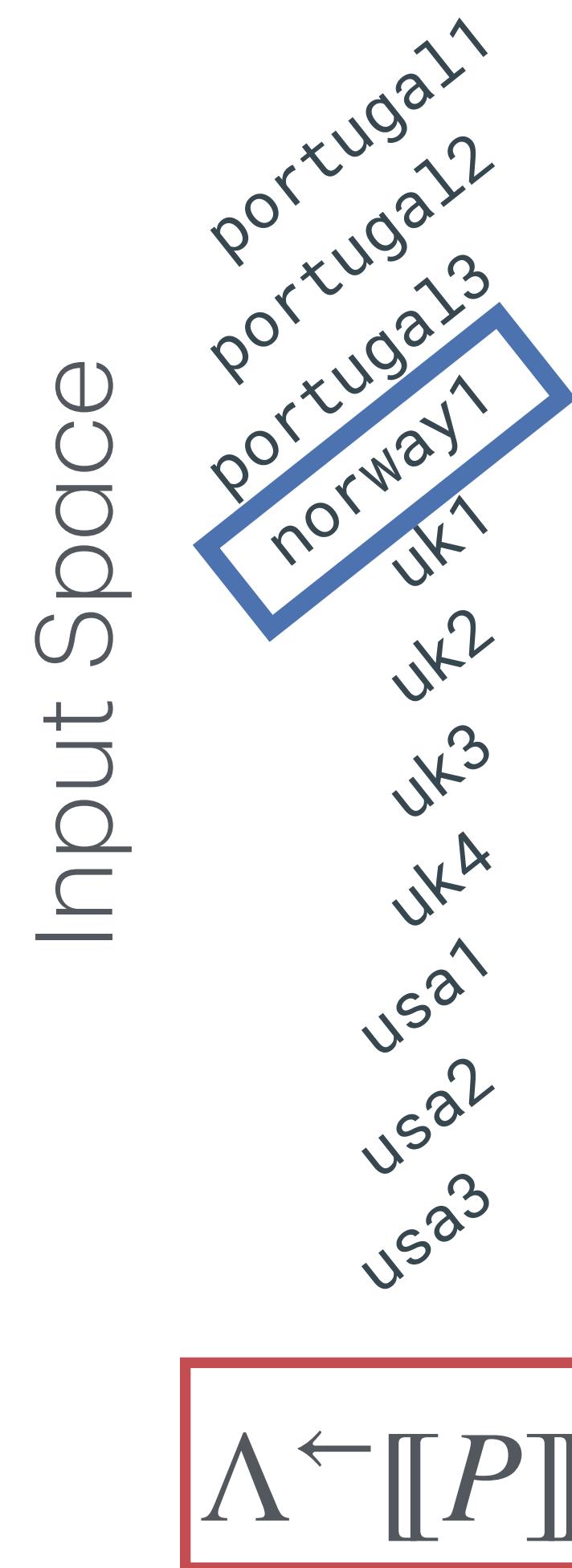
$\Lambda^{\leftarrow} \llbracket P \rrbracket B_1$

$\Lambda^{\leftarrow} \llbracket P \rrbracket B_2$

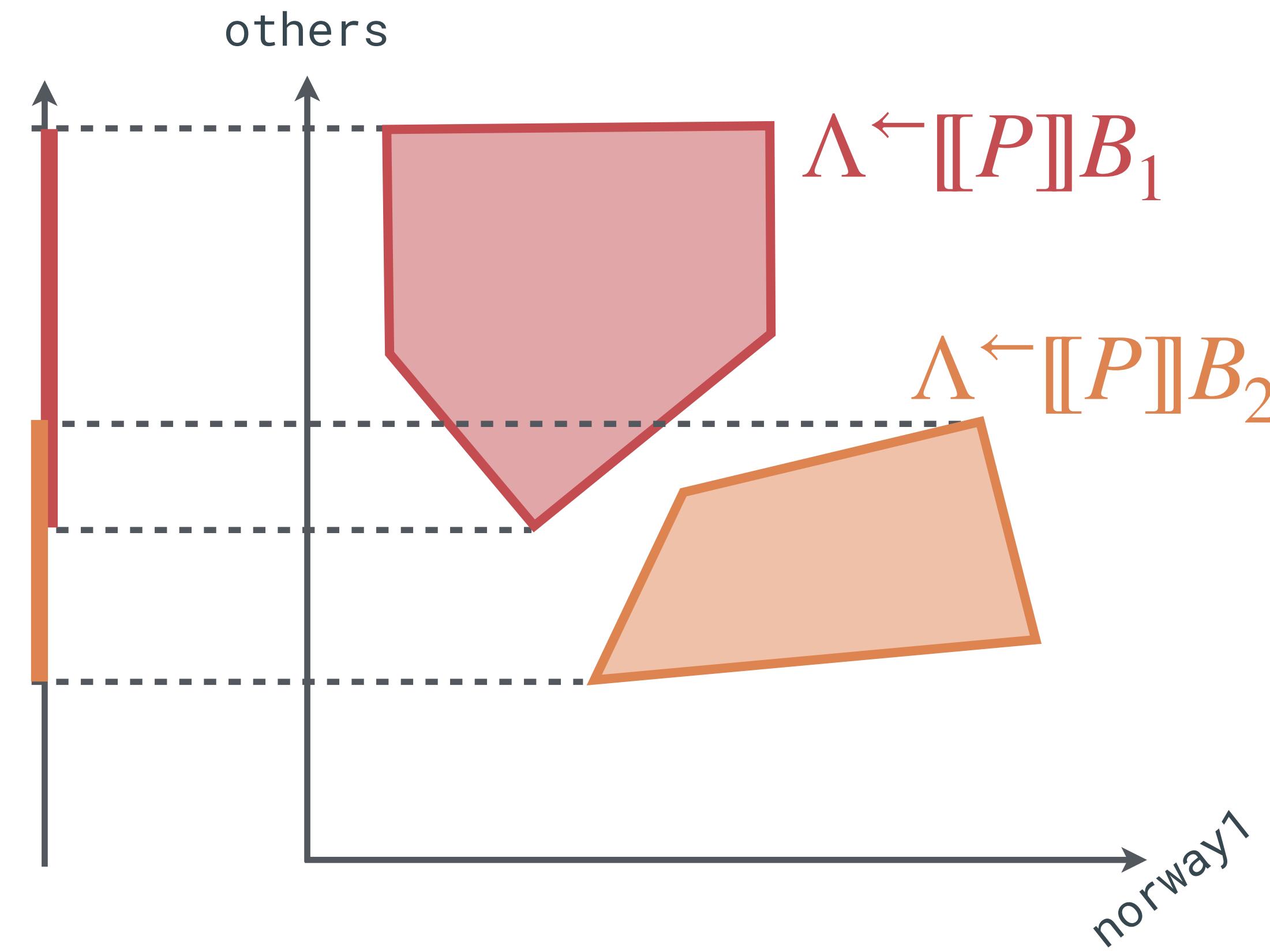
What are the output values that can be computed  
by changes in **norway1** alone?



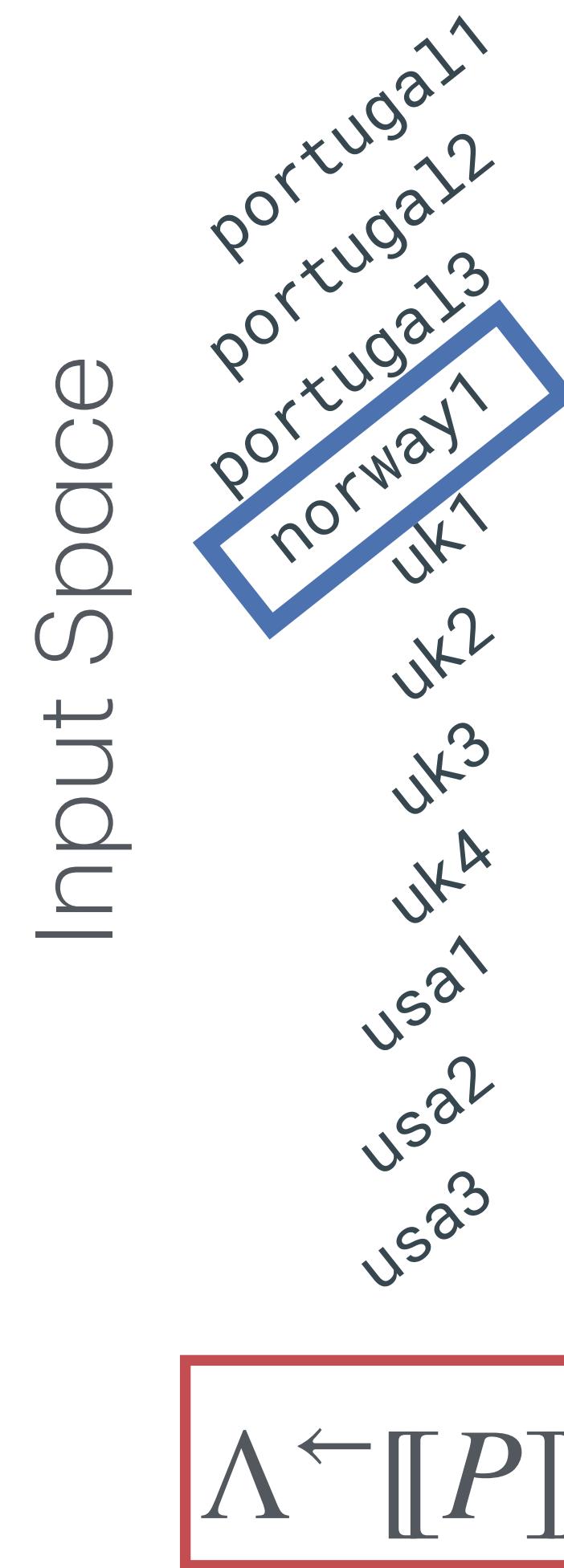
# (iii) Range $^\natural$ Abstract Implementation



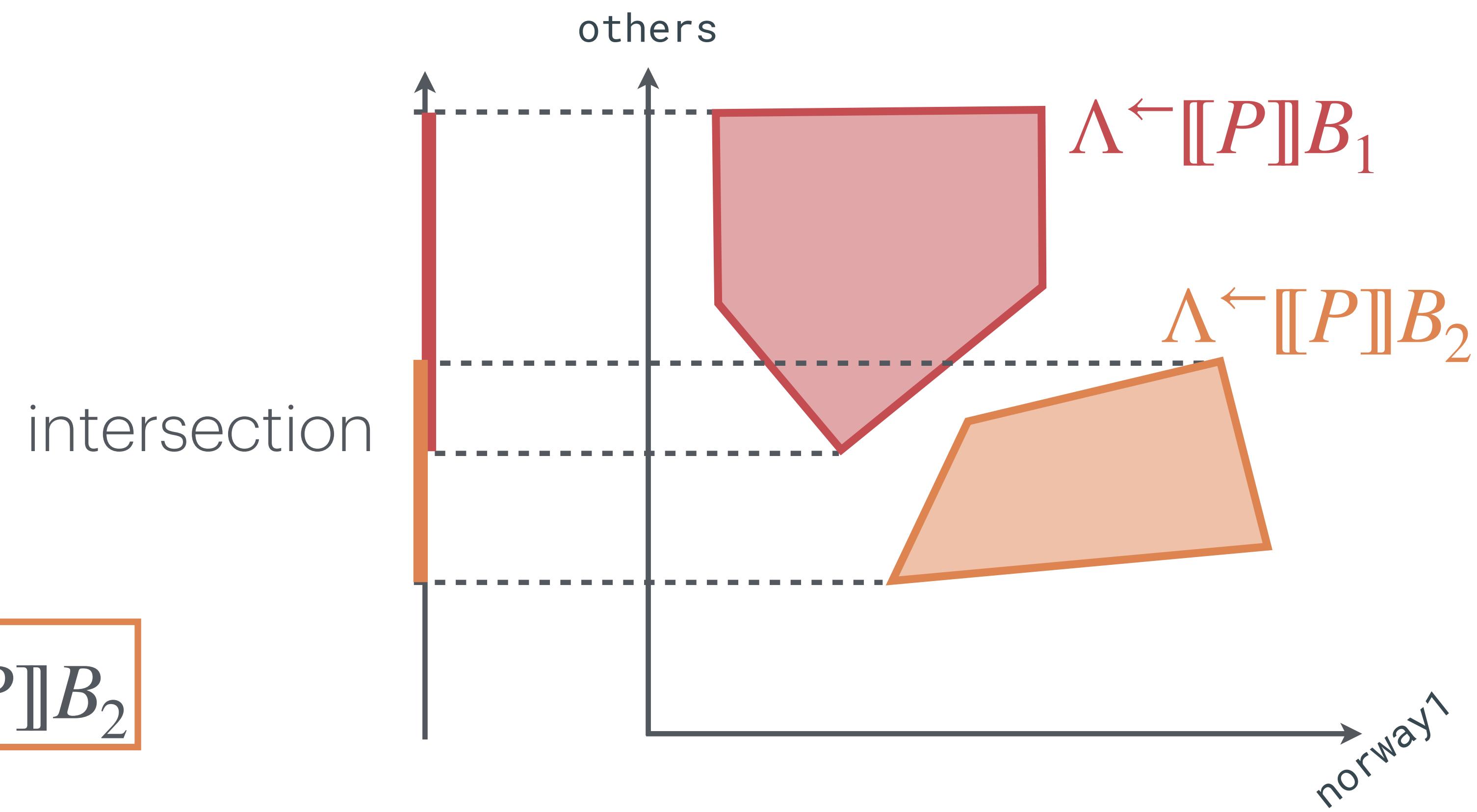
What are the output values that can be computed by changes in **norway1** alone?



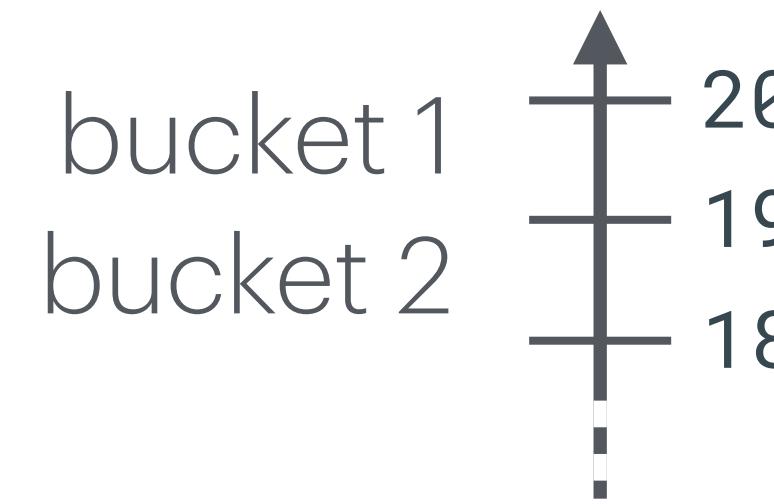
# (iii) Range $\models$ Abstract Implementation



What are the output values that can be computed by changes in `norway1` alone?

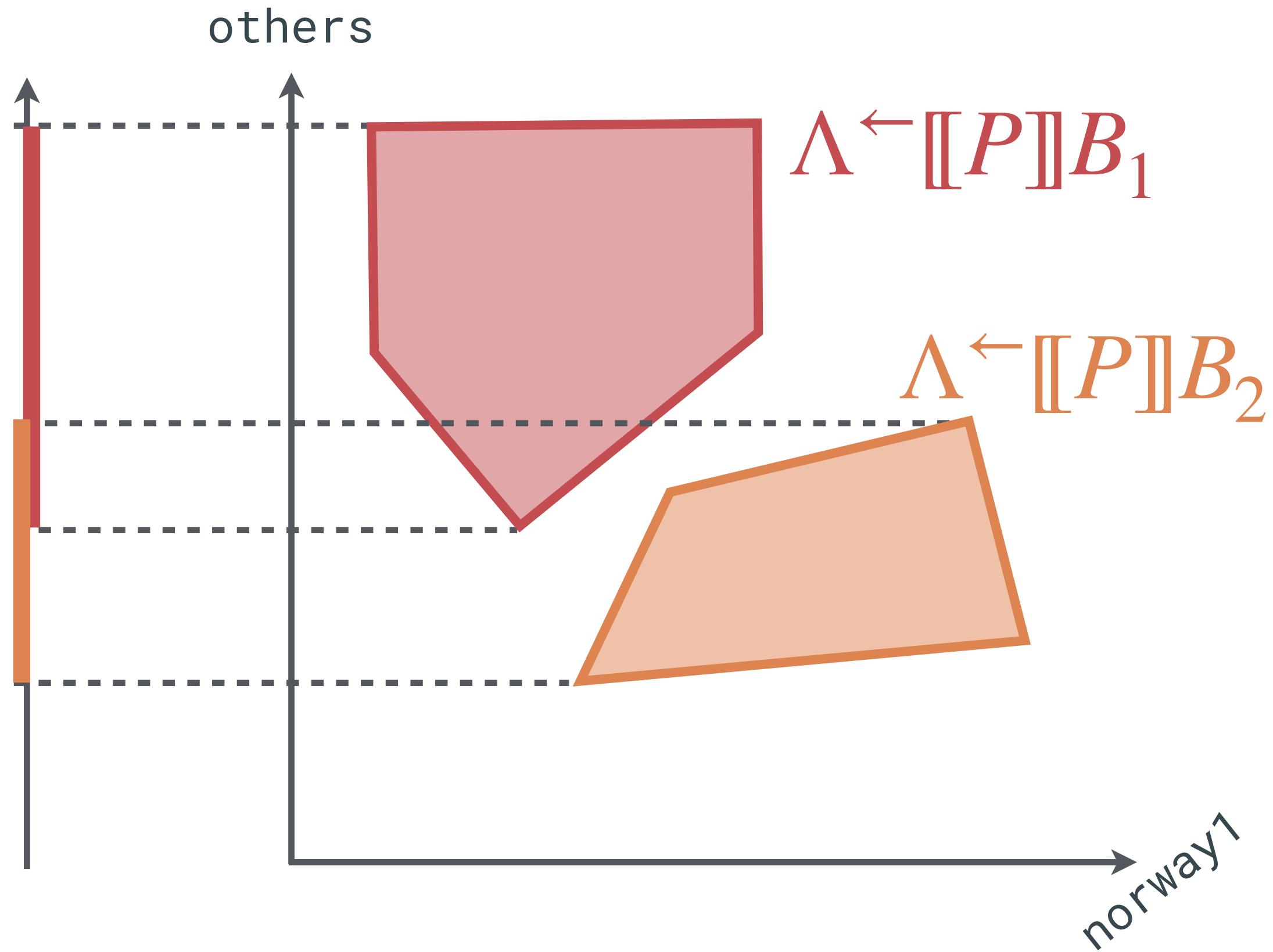


# (iii) Range<sup>♯</sup> Abstract Implementation

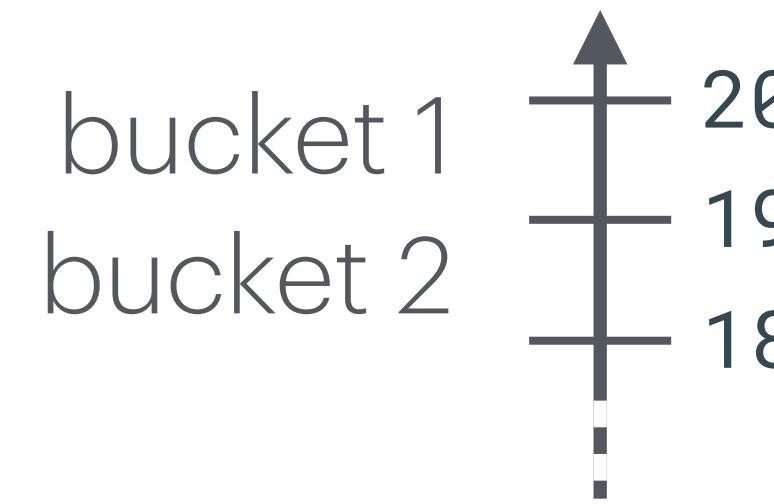


May exist two different  
input values  $x, y$   
differing only in the value of  
**norway1** such that

What are the output values that can be computed  
by changes in **norway1** alone?



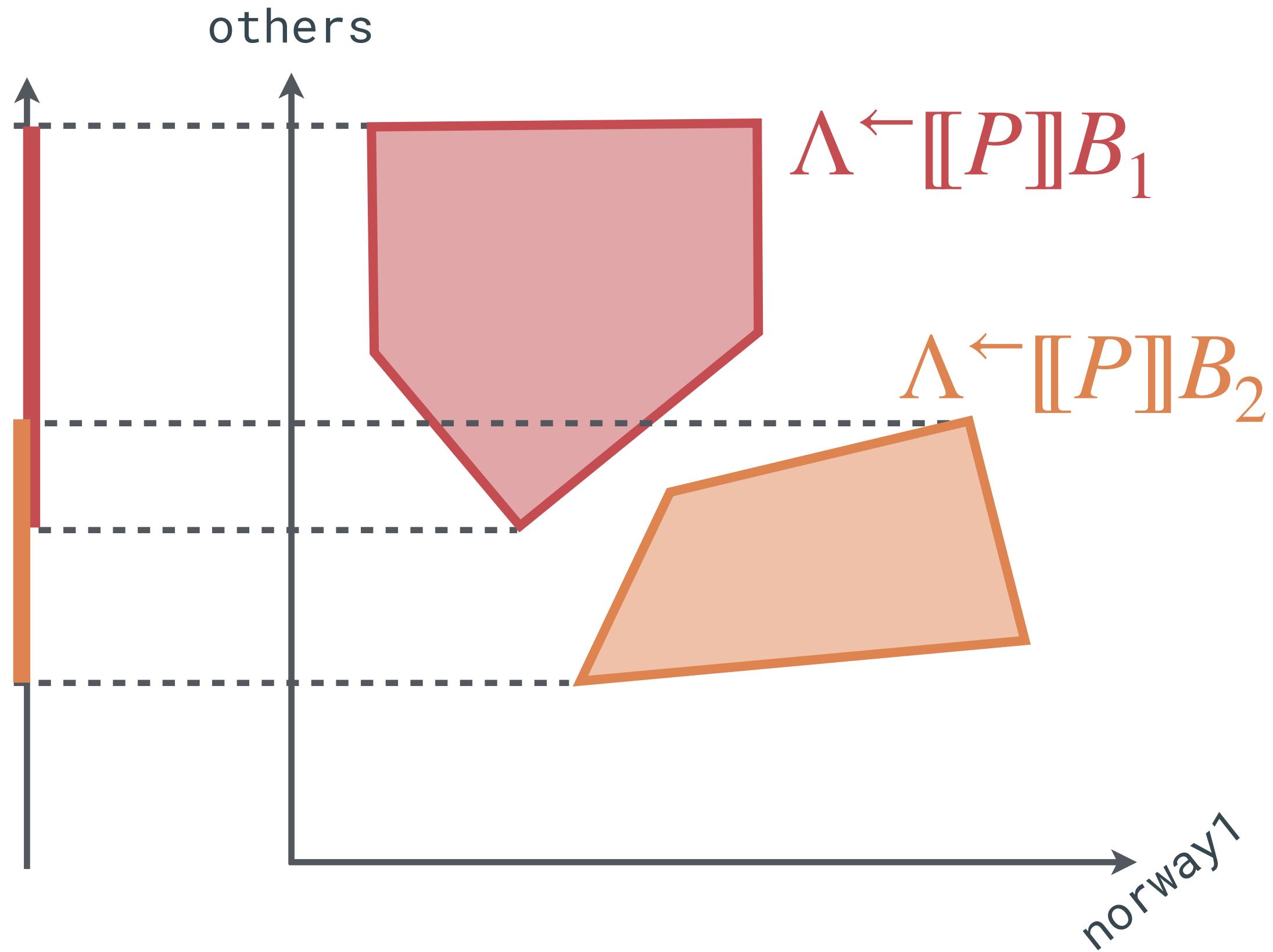
# (iii) Range<sup>♯</sup> Abstract Implementation



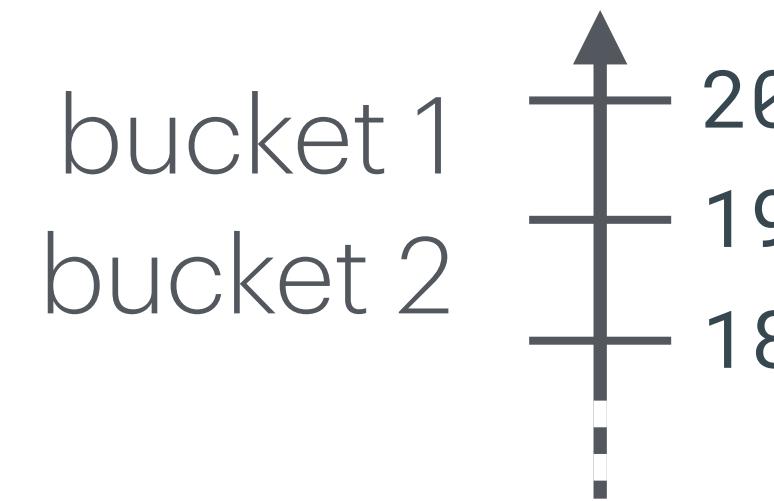
May exist two different  
input values  $x, y$   
differing only in the value of  
`norway1` such that

`mean_growth_rate_60_90(x) ∈ B1`

What are the output values that can be computed  
by changes in `norway1` alone?



# (iii) Range<sup>♯</sup> Abstract Implementation

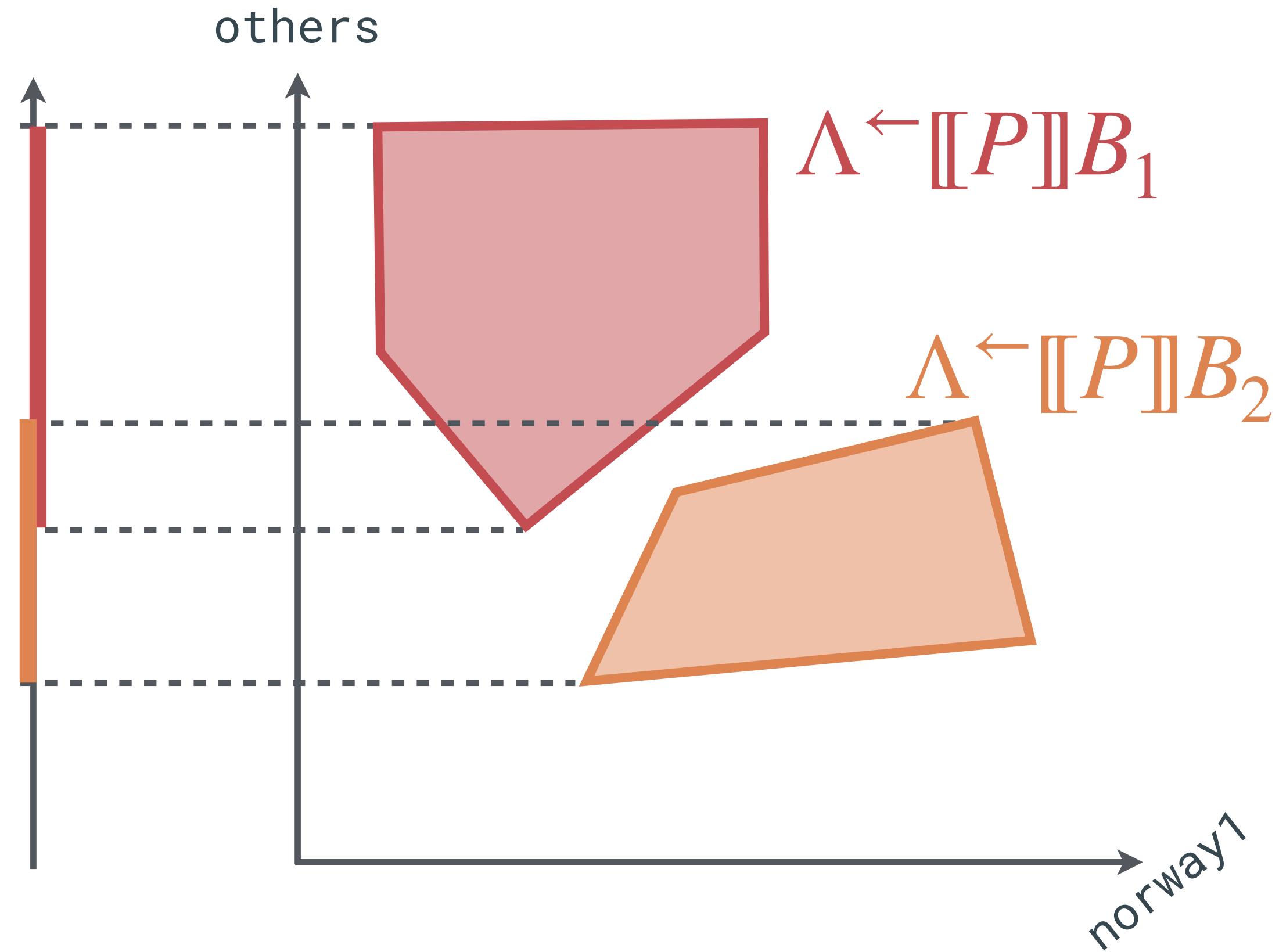


May exist two different  
input values  $x, y$   
differing only in the value of  
`norway1` such that

$$\text{mean\_growth\_rate\_60\_90}(x) \in B_1$$

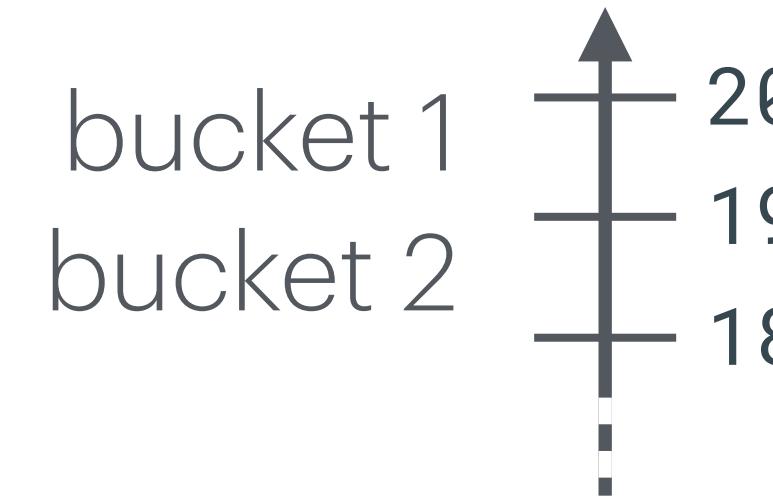
$$\text{mean\_growth\_rate\_60\_90}(y) \in B_2$$

What are the output values that can be computed  
by changes in `norway1` alone?



# (iii) Range<sup>¶</sup> Abstract Implementation

---



What are the output values that can be computed by changes in **norway1** alone?

May exist two different input values  $x, y$  differing only in the value of **norway1** such that

$$\text{mean\_growth\_rate\_60\_90}(x) \in B_1$$

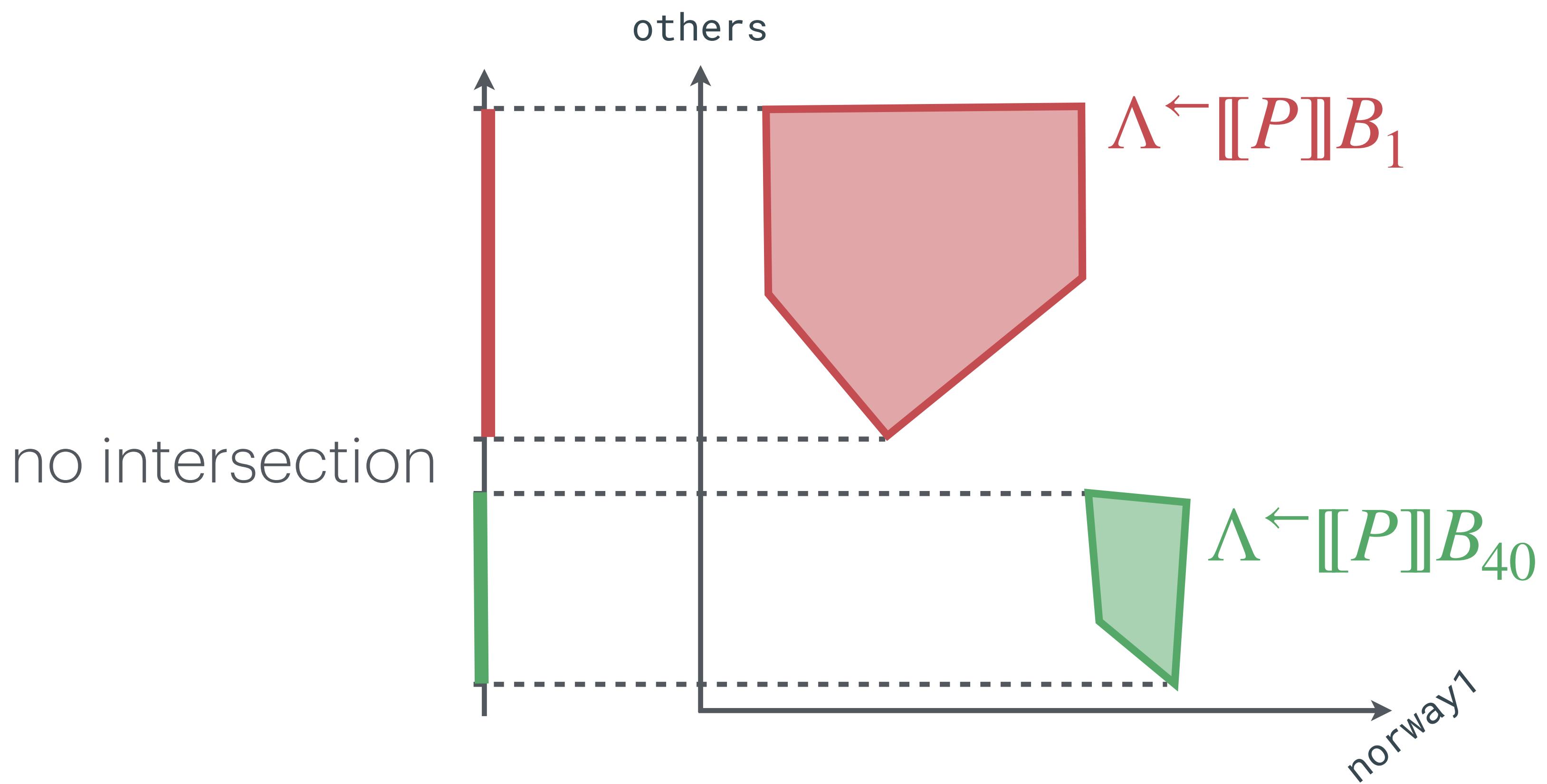
$$\text{mean\_growth\_rate\_60\_90}(y) \in B_2$$

Conclusion:

the impact of **norway1** is at least 2

# (iii) Range<sup>♯</sup> Abstract Implementation

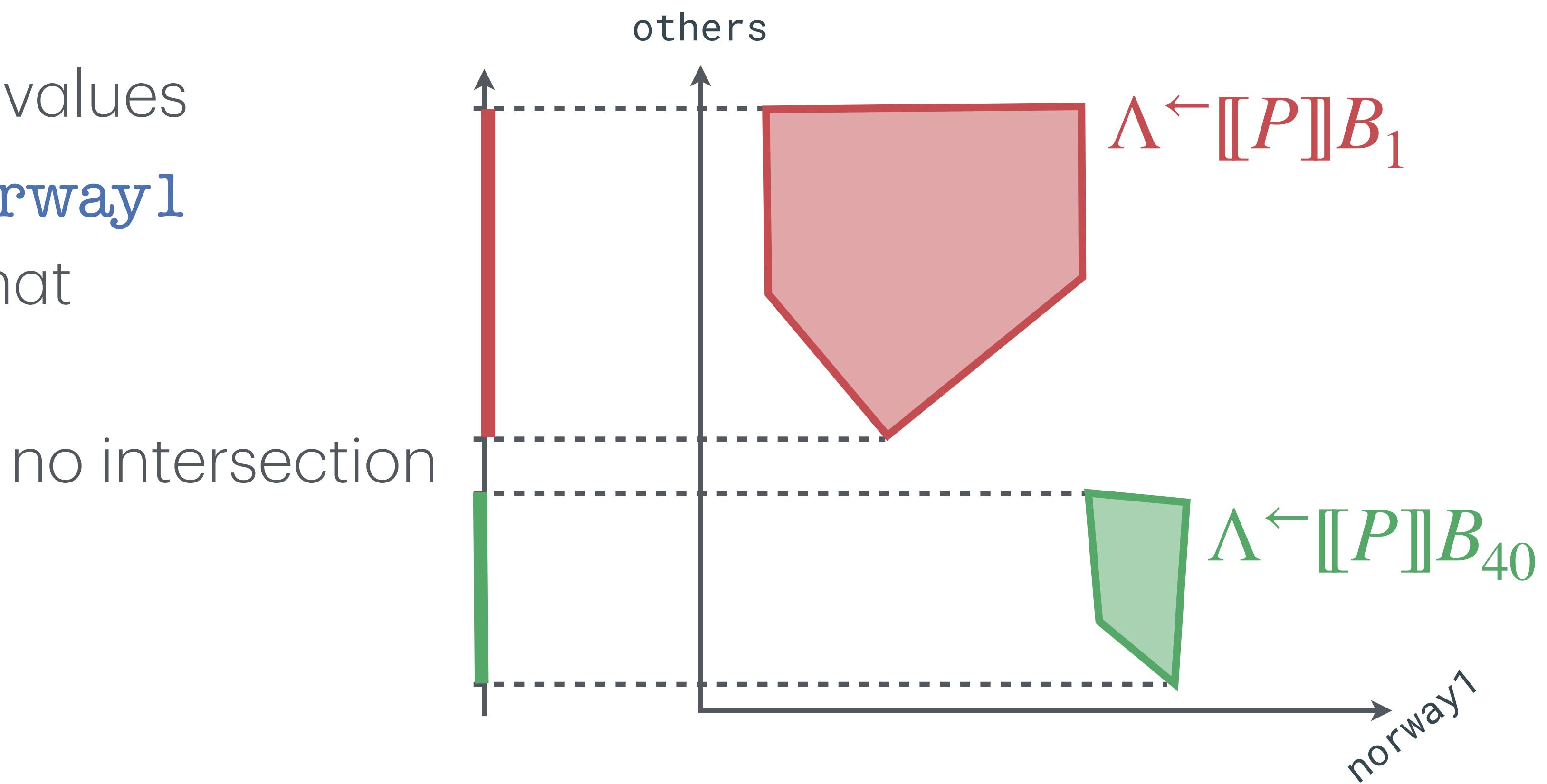
What are the output values that can be computed by changes in `norway1` alone?



# (iii) Range<sup>♯</sup> Abstract Implementation

What are the output values that can be computed by changes in `norway1` alone?

Definitely for any input values  $x, y$  differing only on `norway1` it cannot happen that



# (iii) Range<sup>♯</sup> Abstract Implementation

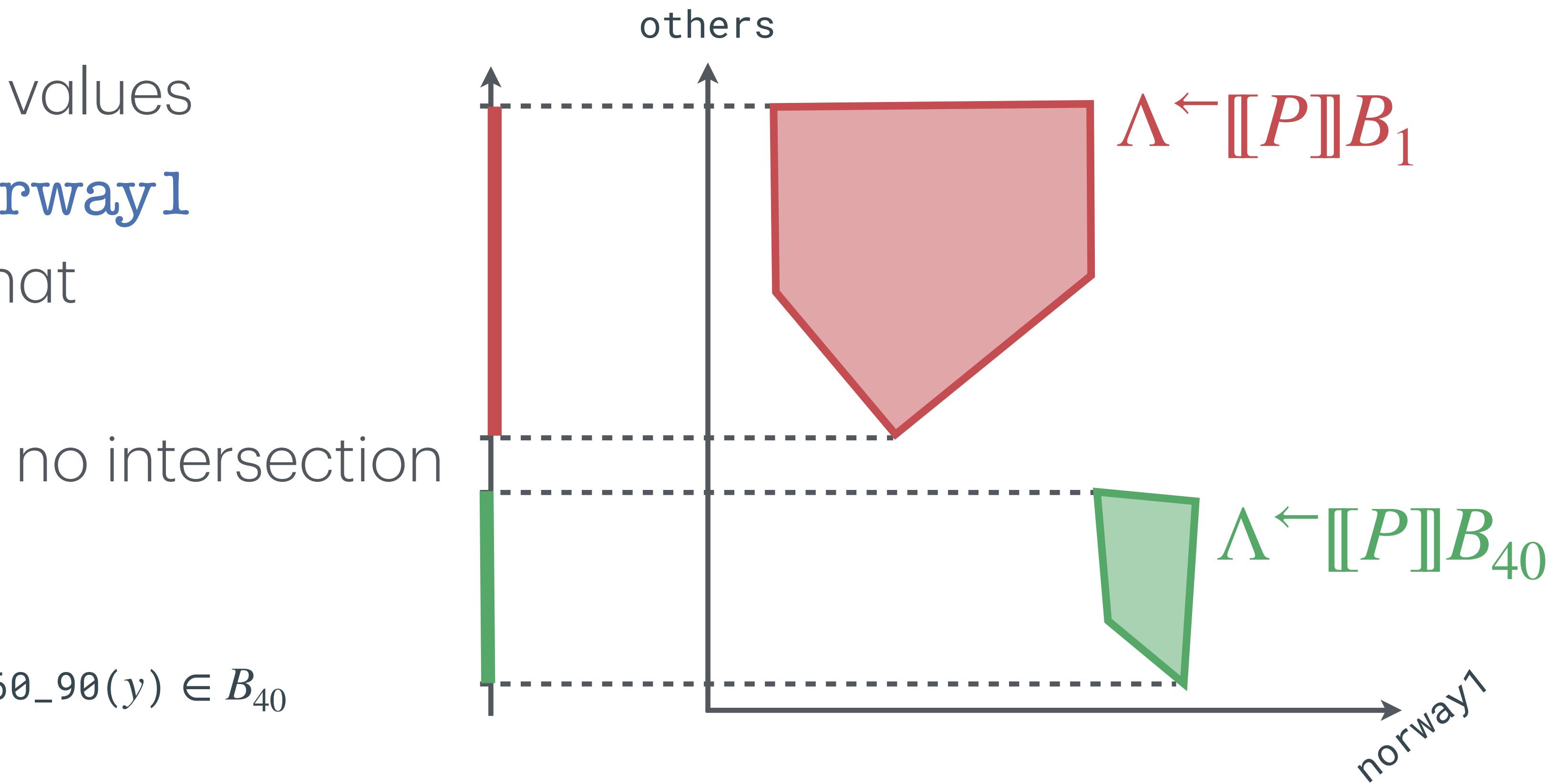
What are the output values that can be computed by changes in `norway1` alone?

Definitely for any input values  $x, y$  differing only on `norway1`  
it cannot happen that

`mean_growth_rate_60_90(x) ∈ B1`

`mean_growth_rate_60_90(y) ∈ B40`

no intersection



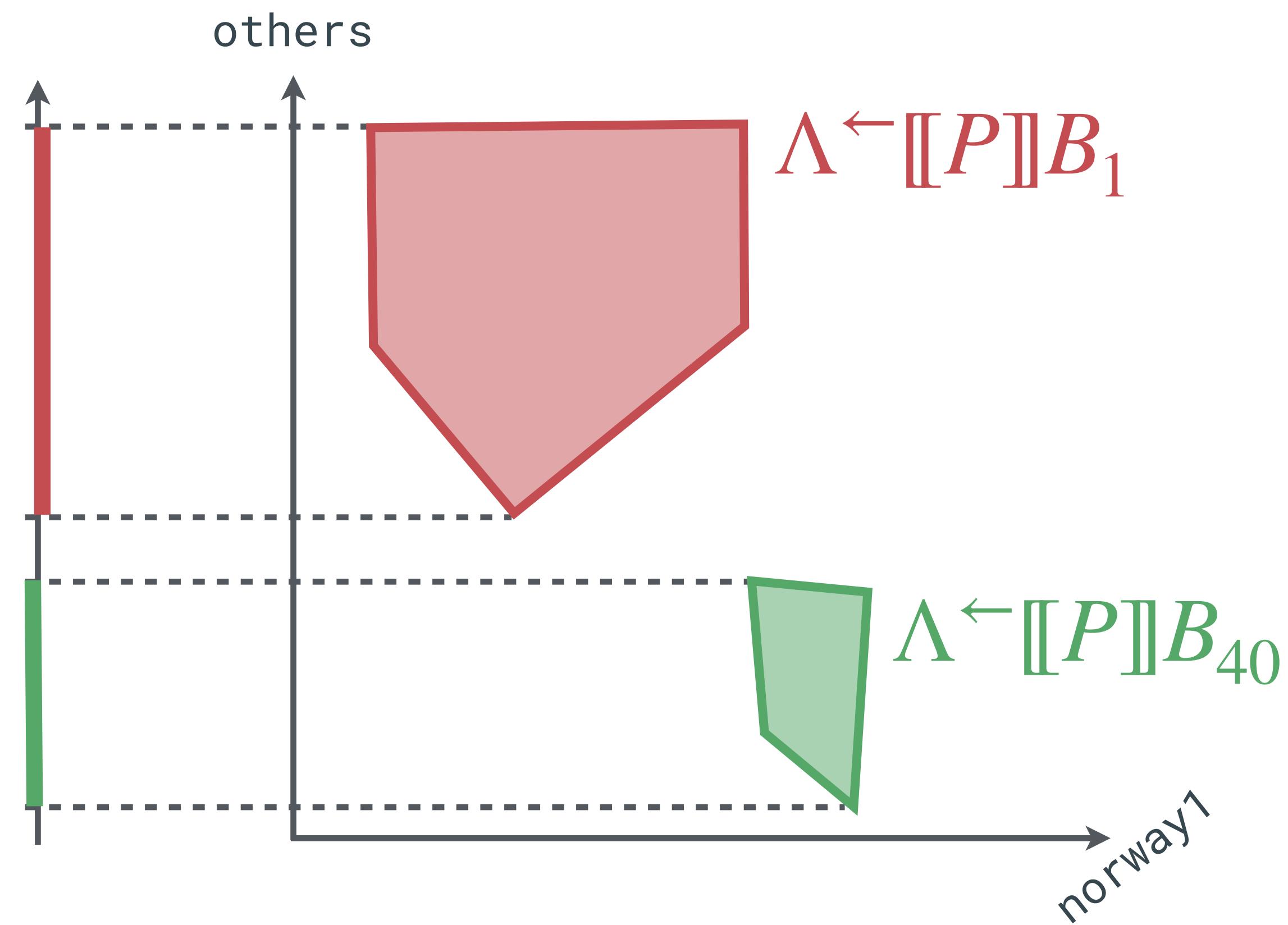
# (iii) Range<sup>♯</sup> Abstract Implementation

What are the output values that can be computed by changes in `norway1` alone?

Definitely for any input values  $x, y$  differing only on `norway1`  
it cannot happen that

$\text{mean\_growth\_rate\_60\_90}(x) \in B_1$   
 $\text{mean\_growth\_rate\_60\_90}(y) \in B_{40}$

no intersection



## (iii) Range<sup>¶</sup> Abstract Implementation

---

What are the output values that can be computed by changes in `norway1` alone?

Definitely for any input values  $x, y$  differing only on `norway1`  
it cannot happen that

Conclusion:

the impact of `norway1` cannot be 40

~~`mean_growth_rate_60_90(x) ∈ B1`~~

~~`mean_growth_rate_60_90(y) ∈ B40`~~

### (iii) Range $^\natural$ Abstract Implementation

---

$P = \text{mean\_growth\_rate\_60\_90}$

Check all possible combinations after projecting away `norway1`

$$\begin{array}{ccccccccc} & \cap & \Lambda^{\leftarrow}[\![P]\!]B_1 & \Lambda^{\leftarrow}[\![P]\!]B_2 & \dots & \Lambda^{\leftarrow}[\![P]\!]B_{40} \\ & \Lambda^{\leftarrow}[\![P]\!]B_1 & \cap & \cap & & & & & \\ & \Lambda^{\leftarrow}[\![P]\!]B_2 & \cap & \cap & & & & & \\ & \dots & & & & & \dots & & \\ & \Lambda^{\leftarrow}[\![P]\!]B_{40} & & & & & & \cap & \end{array}$$

### (iii) Range $^\natural$ Abstract Implementation

$P = \text{mean\_growth\_rate\_60\_90}$

Check all possible combinations after projecting away `norway1`

$\cap$	$\Lambda^{\leftarrow}[\![P]\!]B_1$	$\Lambda^{\leftarrow}[\![P]\!]B_2$	...	$\Lambda^{\leftarrow}[\![P]\!]B_{40}$
$\Lambda^{\leftarrow}[\![P]\!]B_1$	1	2		0
$\Lambda^{\leftarrow}[\![P]\!]B_2$	2	1		0
...			...	
$\Lambda^{\leftarrow}[\![P]\!]B_{40}$	0	0		1

# (iii) Range $^\natural$ Abstract Implementation

$P = \text{mean\_growth\_rate\_60\_90}$

Check all possible combinations after projecting away `norway1`

$\cap$	$\Lambda^{\leftarrow}[\![P]\!]B_1$	$\Lambda^{\leftarrow}[\![P]\!]B_2$	$\dots$	$\Lambda^{\leftarrow}[\![P]\!]B_{17}$	$\dots$	$\Lambda^{\leftarrow}[\![P]\!]B_{40}$
$\Lambda^{\leftarrow}[\![P]\!]B_1$	1	2		0		0
$\Lambda^{\leftarrow}[\![P]\!]B_2$	2	1		0		0
$\dots$			$\dots$		$\dots$	
$\Lambda^{\leftarrow}[\![P]\!]B_7$	7	6		10		0
$\dots$			$\dots$		$\dots$	
$\Lambda^{\leftarrow}[\![P]\!]B_{40}$	0	0		1		1

# (iii) Range $^\natural$ Abstract Implementation

$P = \text{mean\_growth\_rate\_60\_90}$

Check all possible combinations after projecting away `norway1`

$\cap$	$\Lambda^{\leftarrow}[\![P]\!]B_1$	$\Lambda^{\leftarrow}[\![P]\!]B_2$	$\dots$	$\Lambda^{\leftarrow}[\![P]\!]B_{17}$	$\dots$	$\Lambda^{\leftarrow}[\![P]\!]B_{40}$
$\Lambda^{\leftarrow}[\![P]\!]B_1$	1	2		0		0
$\Lambda^{\leftarrow}[\![P]\!]B_2$	2	1		0		0
$\dots$			$\dots$		$\dots$	
$\Lambda^{\leftarrow}[\![P]\!]B_7$	7	6		10		0
$\dots$			$\dots$		$\dots$	
$\Lambda^{\leftarrow}[\![P]\!]B_{40}$	0	0		1		1

# (iii) Range $^\natural$ Abstract Implementation

$P = \text{mean\_growth\_rate\_60\_90}$

Check all possible combinations after projecting away `norway1`

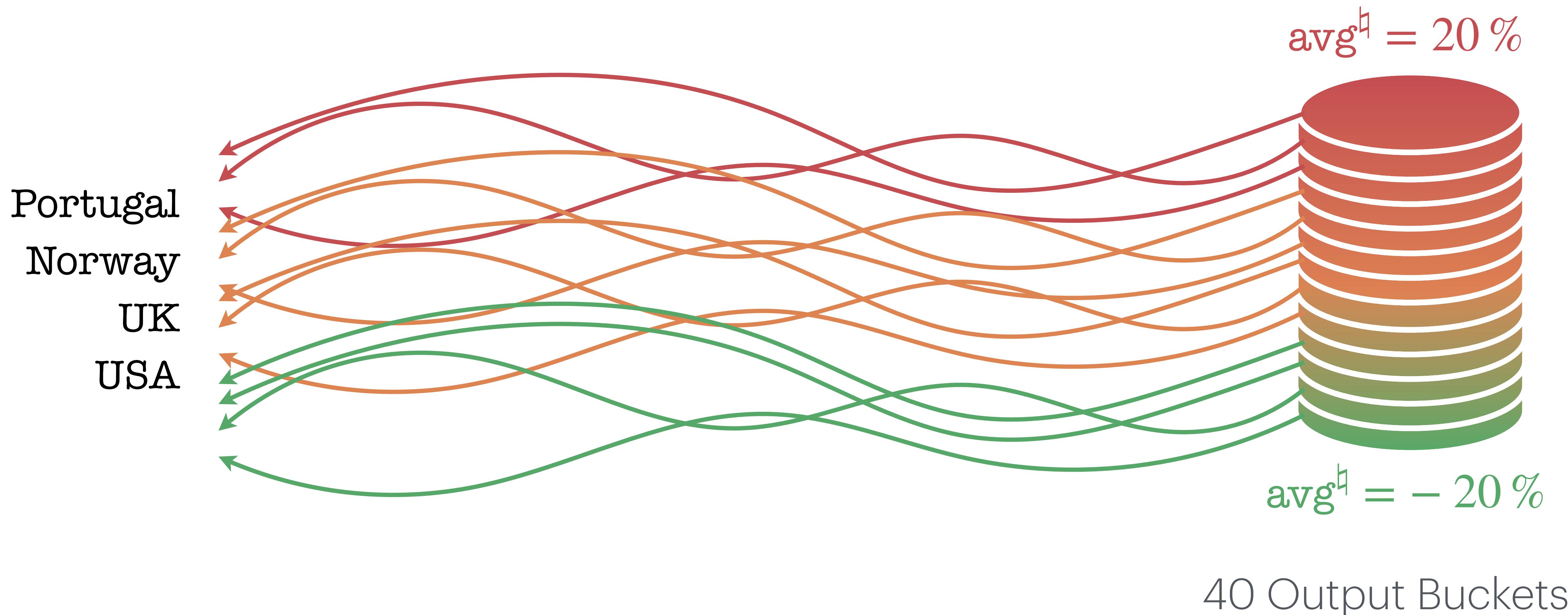
$\cap$	$\Lambda^{\leftarrow}[\![P]\!]B_1$	$\Lambda^{\leftarrow}[\![P]\!]B_2$	$\dots$	$\Lambda^{\leftarrow}[\![P]\!]B_{17}$	$\dots$	$\Lambda^{\leftarrow}[\![P]\!]B_{40}$	
$\Lambda^{\leftarrow}[\![P]\!]B_1$	1	2		0		0	maximum
$\Lambda^{\leftarrow}[\![P]\!]B_2$	2	1		0		0	
$\dots$			$\dots$		$\dots$		
$\Lambda^{\leftarrow}[\![P]\!]B_7$	7	6		10		0	10
$\dots$			$\dots$		$\dots$		
$\Lambda^{\leftarrow}[\![P]\!]B_{40}$	0	0		1		1	

# (iii) Range<sup>§</sup> Abstract Implementation

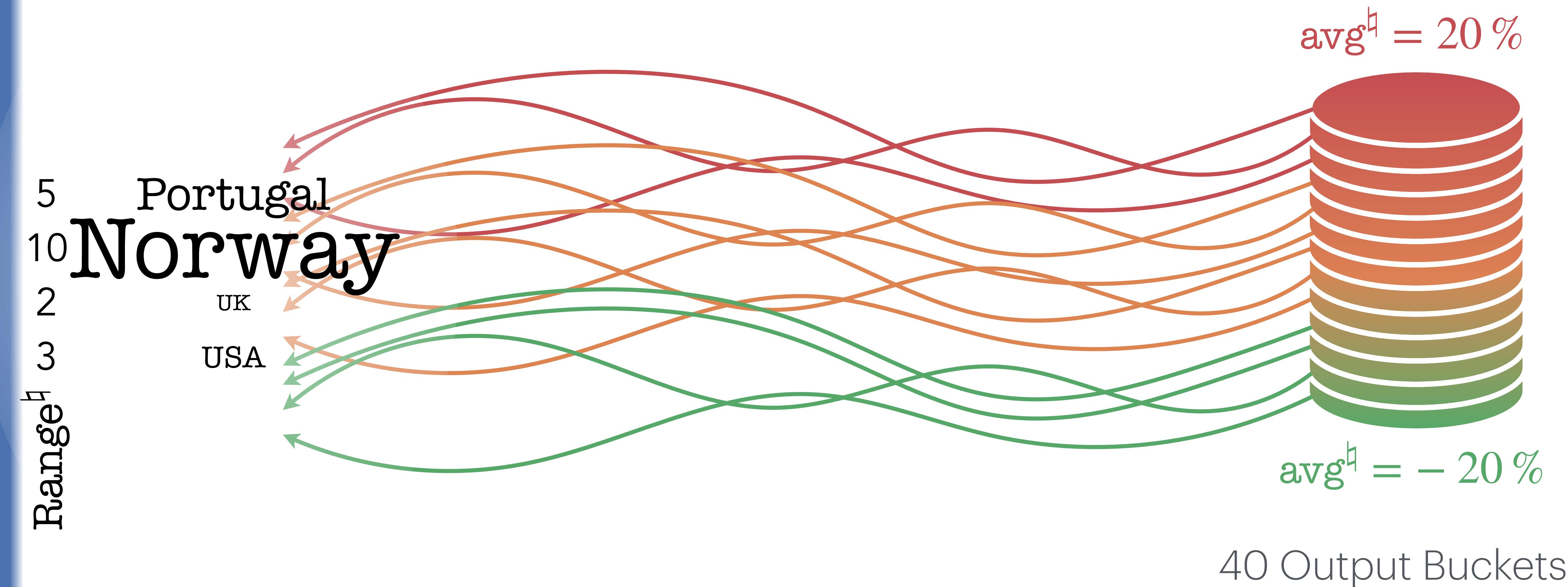
---

The impact of `norway1`  
on the computation of `mean_growth_rate_60_90`  
is at most `10`

# Reinhart and Rogoff



# Reinhart and Rogoff



# Reinhart and Rogoff

---

```
1: def mean_growth_rate_60_90(  
2:     portugal1, portugal2, portugal3,  
3:     norway1,  
4:     uk1, uk2, uk3, uk4,  
5:     usa1, usa2, usa3):  
6:     portugal_avg = avg(portugal1, portugal2, portugal3)  
7:     norway_avg = avg(norway1)  
8:     uk_avg = avg(uk1, uk2, uk3, uk4)  
9:     usa_avg = avg(usa1, usa2, usa3)  
10:    return avg(portugal_avg, norway_avg, uk_avg, usa_avg)
```

# Reinhart and Rogoff

---

```
1: def mean_growth_rate_60_90(  
2:     portugal1, portugal2, portugal3,  
3:     norway1,  
4:     uk1, uk2, uk3, uk4,  
5:     usa1, usa2, usa3):  
6:     portugal_avg = avg(portugal1, portugal2, portugal3)  
7:     norway_avg = avg(norway1)  
8:     uk_avg = avg(uk1, uk2, uk3, uk4)  
9:     usa_avg = avg(usa1, usa2, usa3)  
10:    return avg(portugal_avg, norway_avg, uk_avg, usa_avg)
```



# Theoretical Framework

## Quantitative Input Usage Static Analysis

Denis Mazzucato, Marco Campion, and Caterina Urban  
NFM 2024

**Quantitative Input Usage Static Analysis**

Denis Mazzucato<sup>[0000-0002-3613-2035]</sup>, Marco Campion<sup>[0000-0002-1099-3494]</sup>,  
and Caterina Urban<sup>[0000-0002-8127-9642]</sup>

INRIA & ENS | PSL,  
`{denis.mazzucato,marco.campion,caterina.urban}@inria.fr`

**Abstract.** Programming errors in software applications may produce plausible yet erroneous results, without providing a clear indication of failure. This happens, for instance, when certain inputs have a disproportionate impact on the program result. To address this issue, we propose a novel quantitative static analysis for determining the impact of inputs on the program computations, parametrized in the definition of impact. This static analysis employs an underlying abstract backward analyzer and computes a sound over-approximation of the impact of program inputs, providing valuable insights into how the analyzed program handles them. We implement a proof-of-concept static analyzer to demonstrate potential applications.

**1 Introduction**

Disastrous outcomes may result from programming errors in safety-critical settings, especially when they do not result in software failures but instead produce a plausible yet erroneous outcome. Such bugs are hard to spot since they provide no indication that something went wrong. A potential source of such errors is when an input variable has disproportionate impact on the program computations compared to the developers' expectations. A notable example is the Reinhart and Rogoff article "Growth in a Time of Debt" [19], which was heavily cited to justify austerity measures around the world in the following years, and was later discovered to be flawed [13]. Notably, one of the several programming and methodological errors discovered in the article is the incorrect usage of the input value relative to Norway's economic growth in 1964, compromising the authors' conclusion. Hence, it is important to employ techniques that enhance the confidence in the usage of input variables.

In this direction, Barowy et al. [2] proposed a stochastic approach specific for spreadsheet applications. Such approach is able to estimate the impact of input cells. However, the lack of mathematical guarantees precludes the employment of such technique in safety-critical contexts. On the other hand, existing formal methods-based approaches only target qualitative properties about input data usage, e.g., only addressing whether an input variable is used or not [22, 23].

In this work, we present a novel quantitative input usage framework to discriminate between input variables with different impact on the outcome of a

# Theoretical Framework

---

## Extensional Properties

---

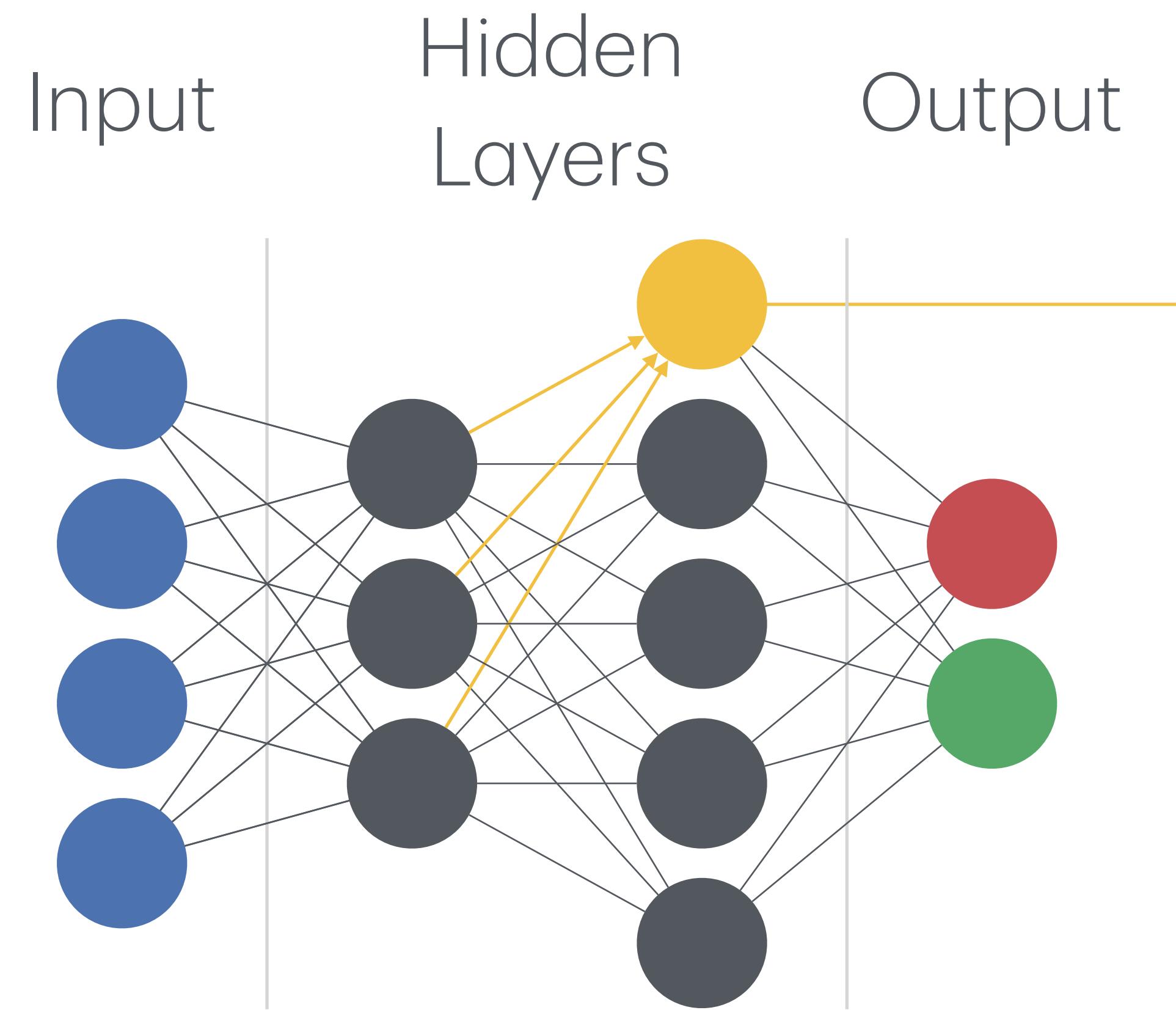
Neural  
Networks

## Intensional Properties

---

Loop  
Iterations

# Neural Networks



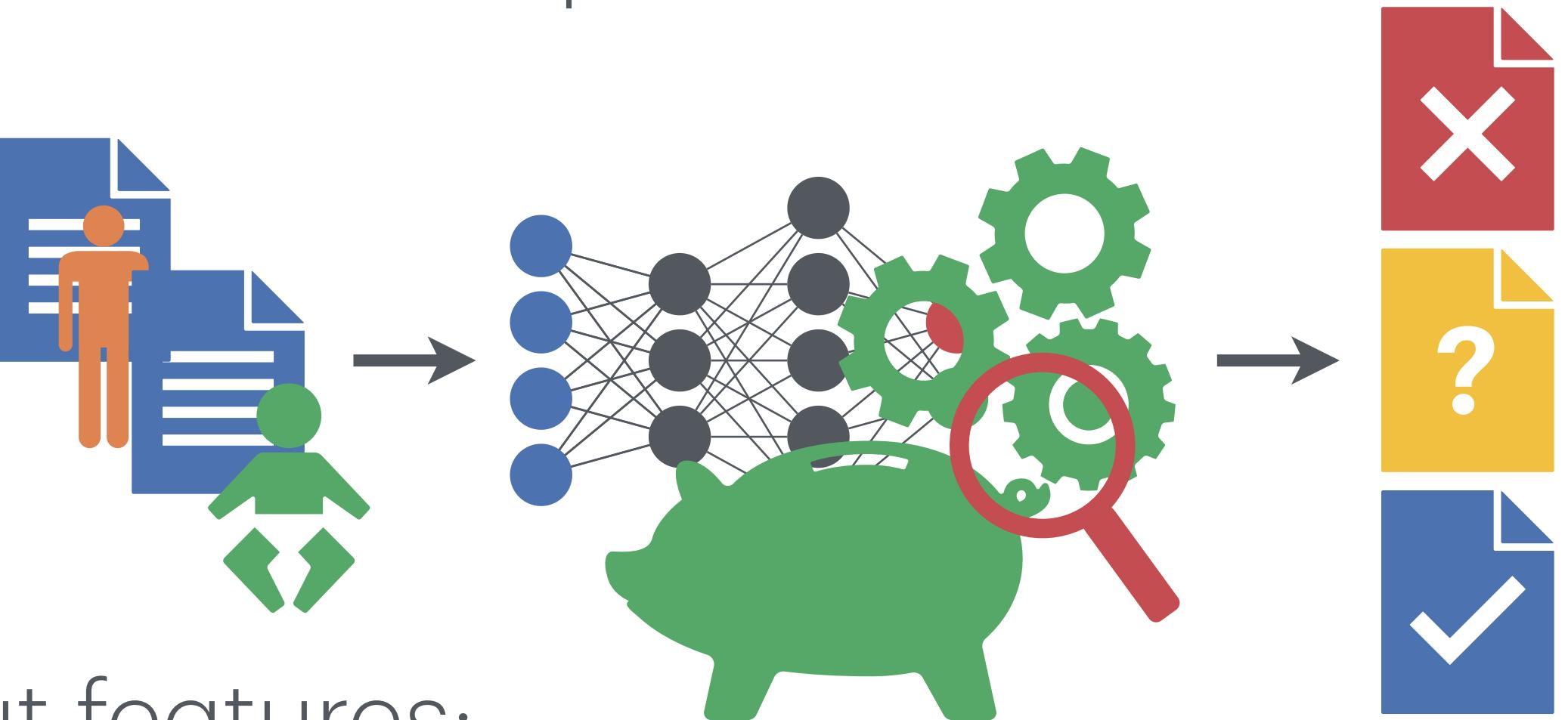
$$x_{i,j} = \text{ReLU}\left(\sum_k w_k \cdot x_{i-1,k} + b_{i,j}\right)$$

$$\max(0, \sum_k w_k \cdot x_{i-1,k} + b_{i,j})$$

$$\hat{x} = \underbrace{\sum_k w_k \cdot x_{i-1,k}}_{\text{intermediate sum}} + b_{i,j}$$

# Quantifying the Fair Input Space

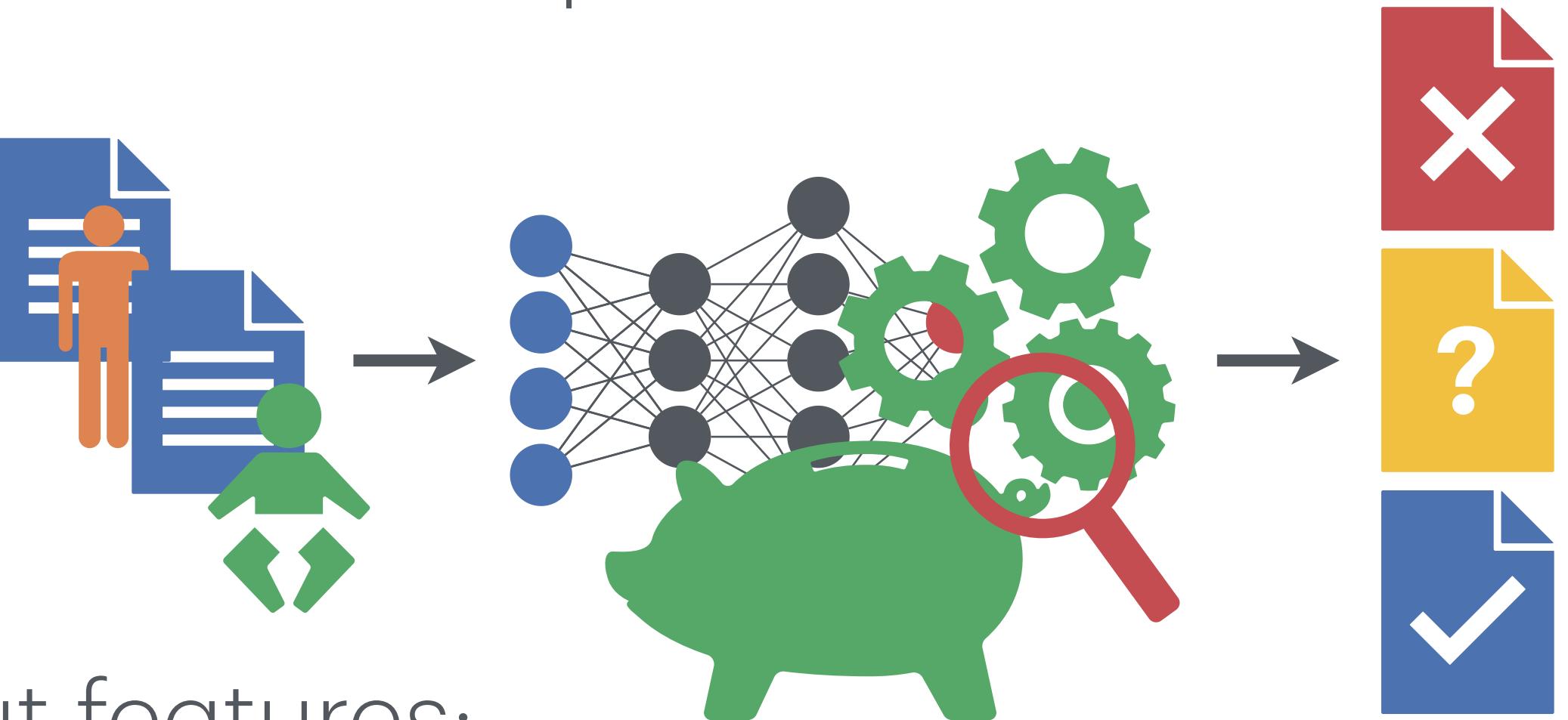
Credit Request Network



Input features:  
amount, age

# Quantifying the Fair Input Space

Credit Request Network



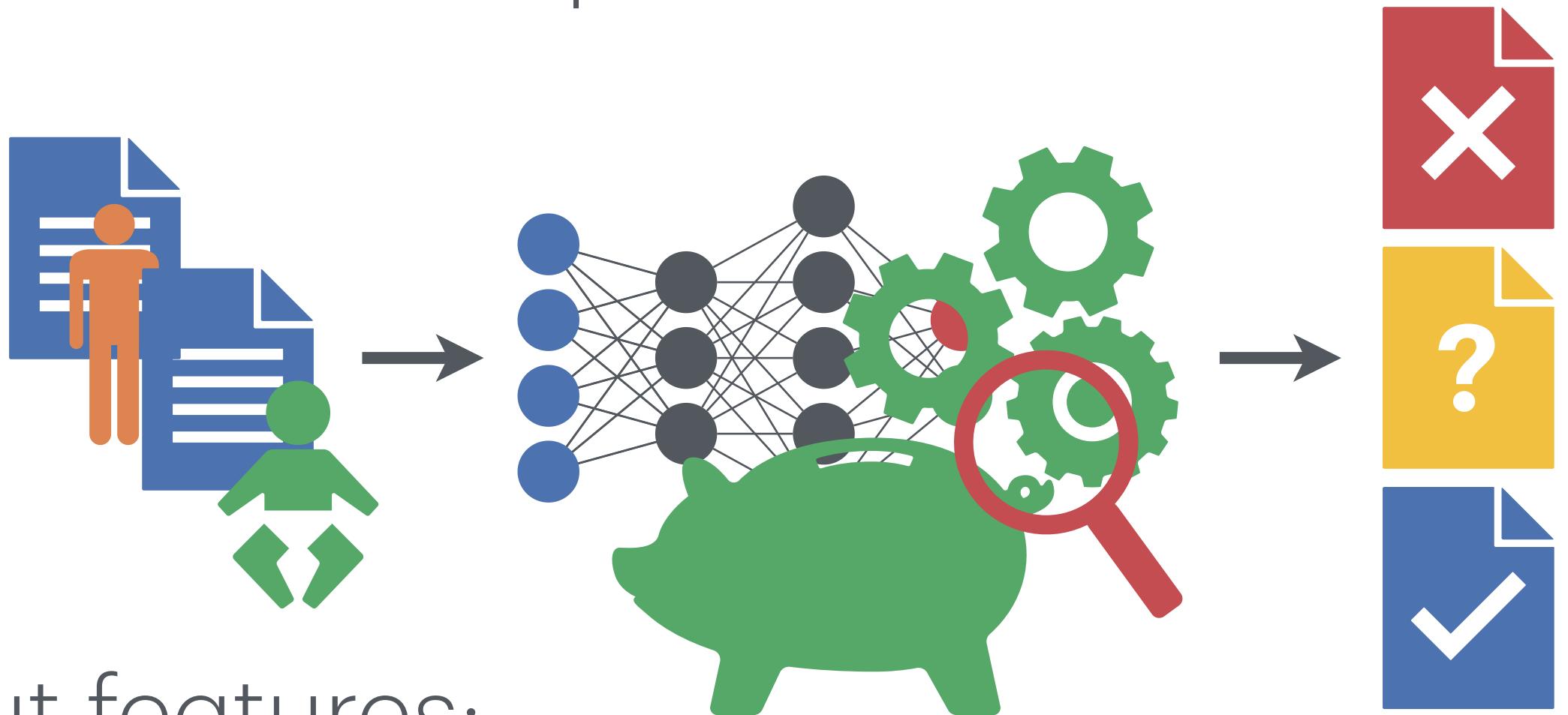
Input features:  
amount, age

Fair = no discrimination

# Quantifying the Fair Input Space

New quantifiers: **QAUNUSED**

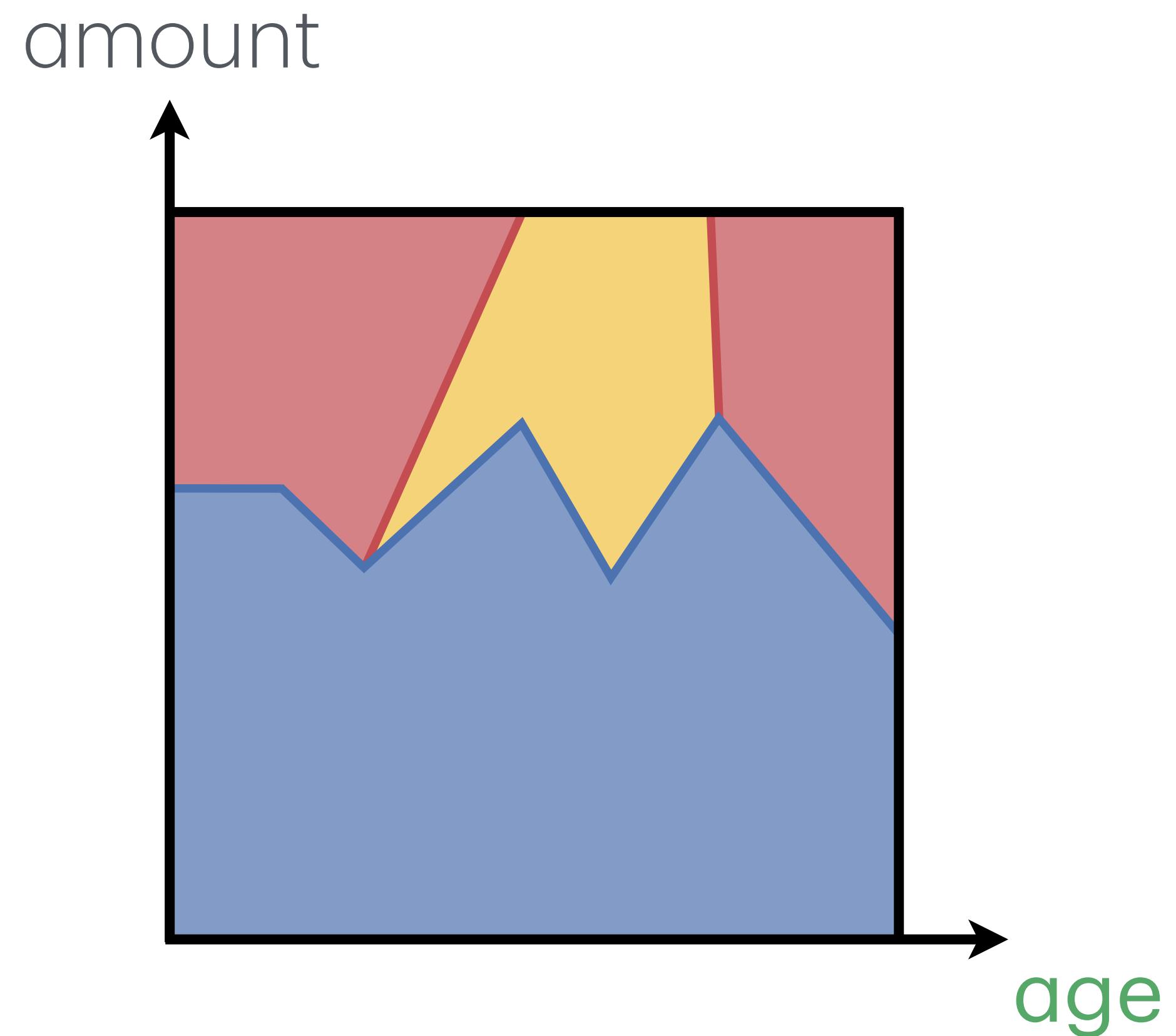
Credit Request Network



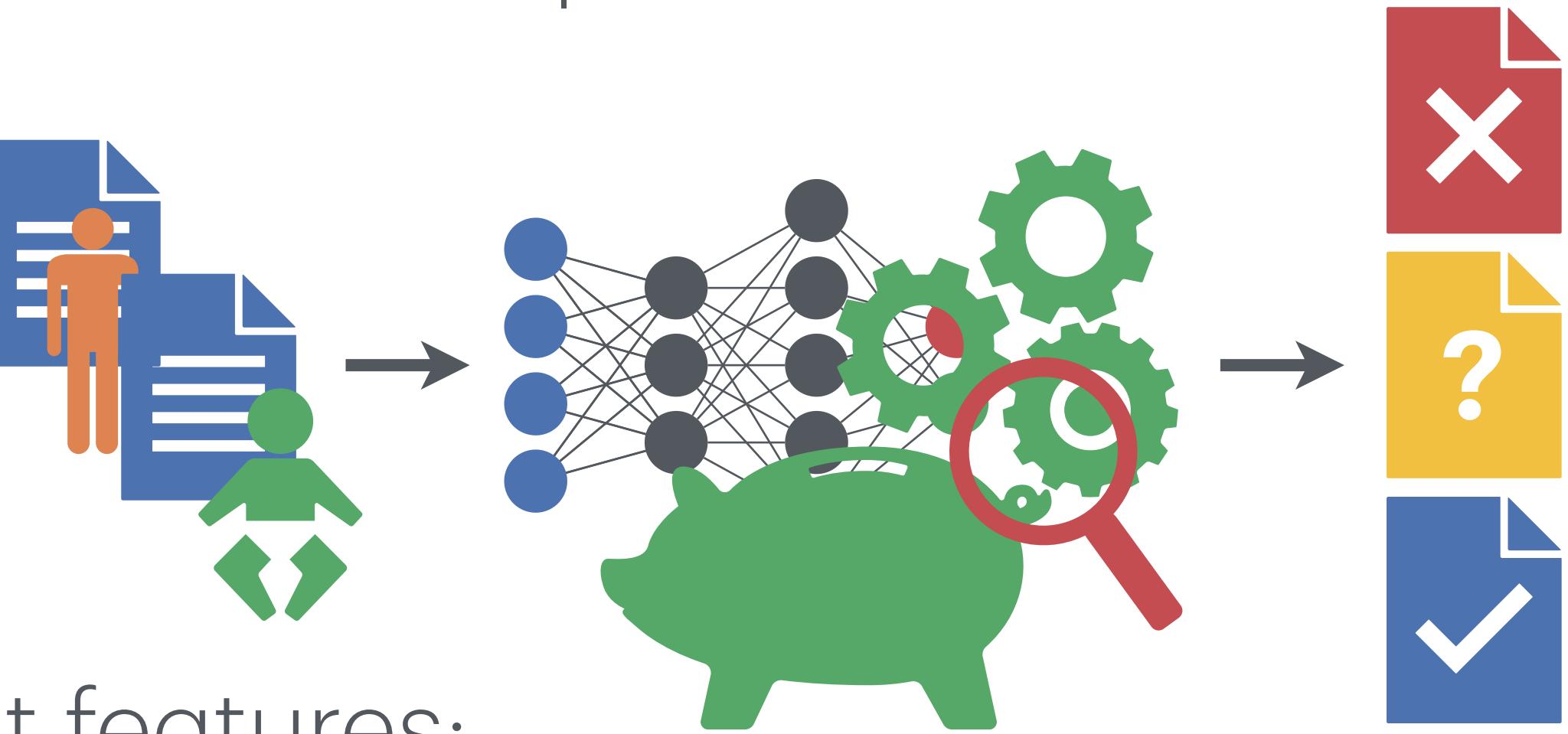
Input features:  
amount, **age**

Fair = no discrimination

# Quantifying the Fair Input Space: QAUNUSED

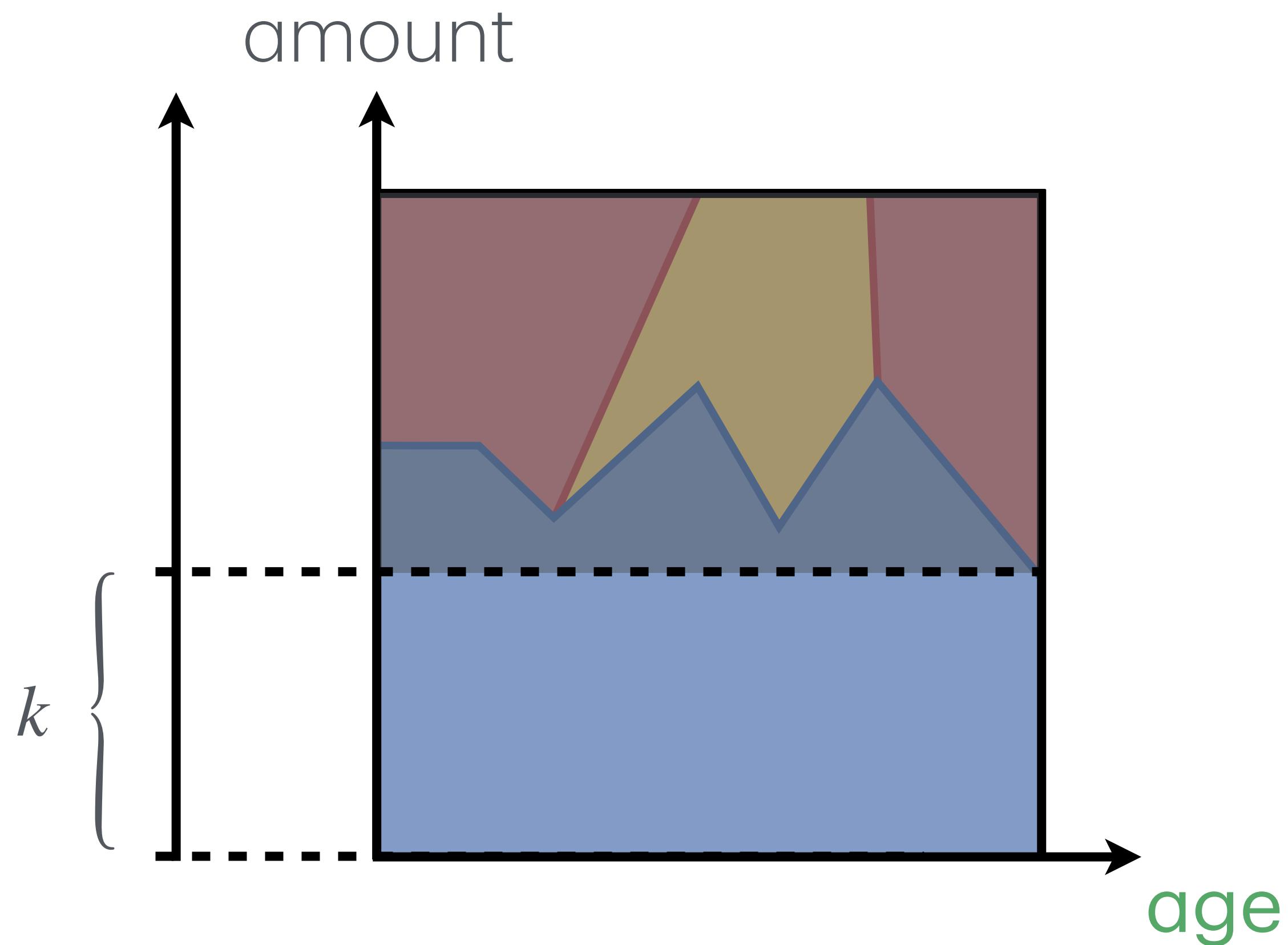


Credit Request Network

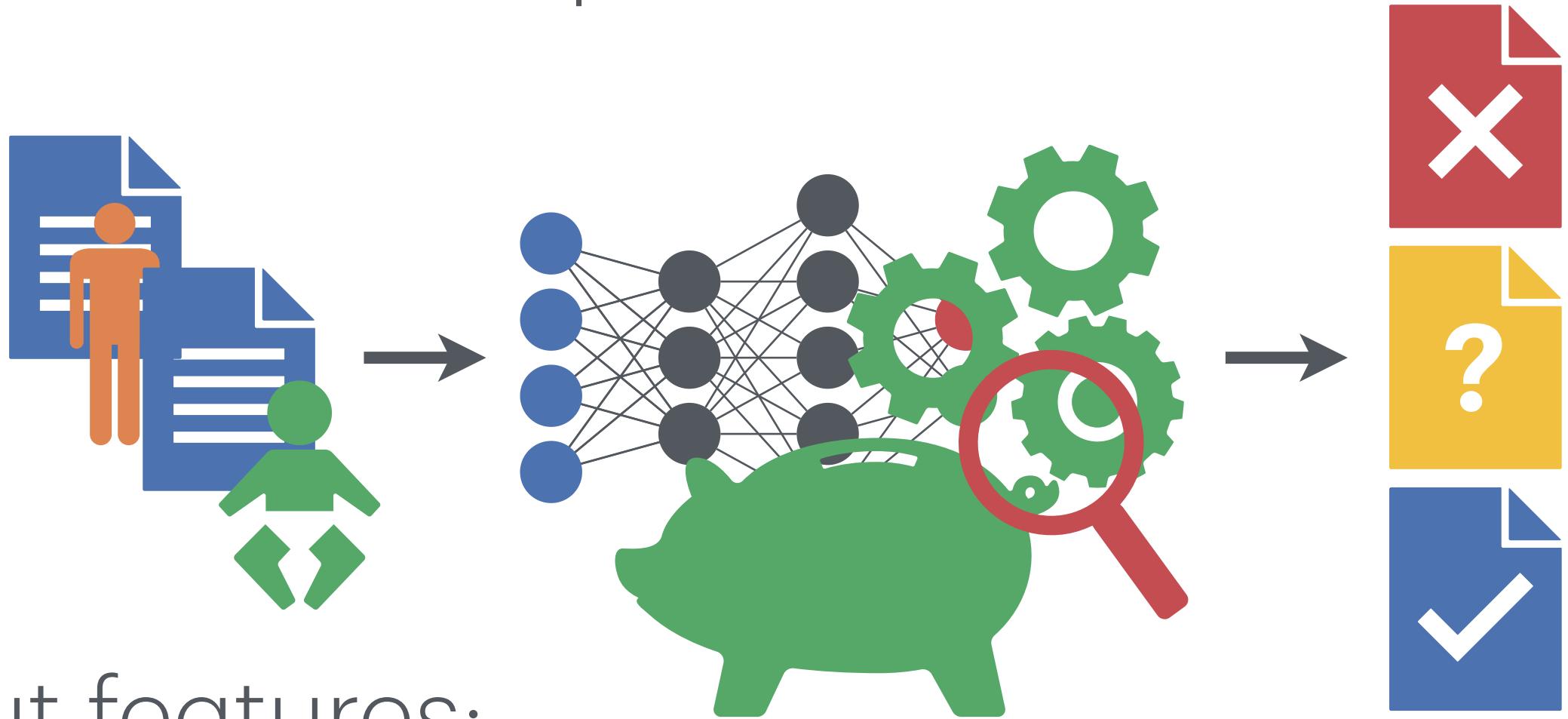


Fair = no discrimination

# Quantifying the Fair Input Space: QAUNUSED

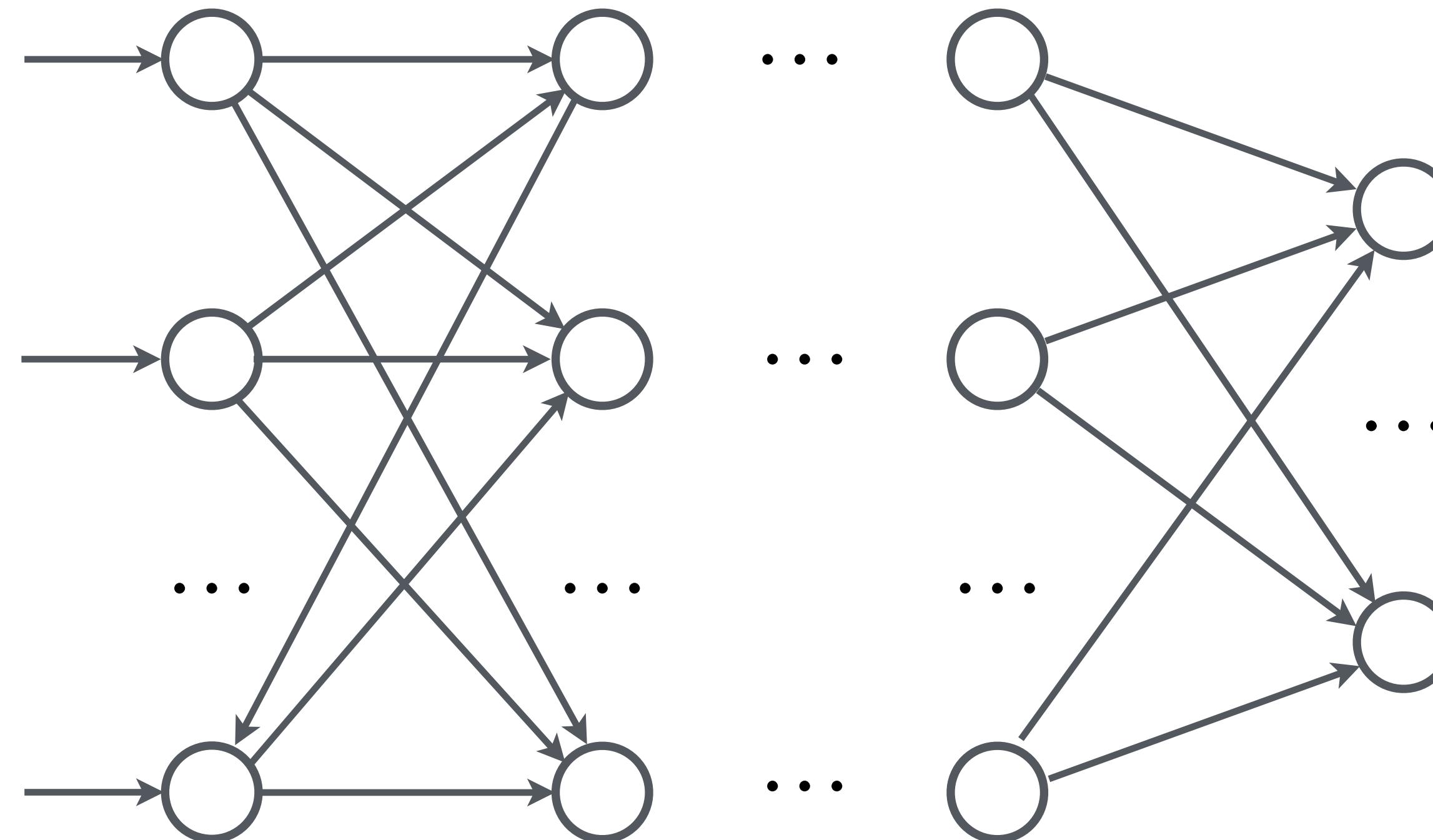


Credit Request Network

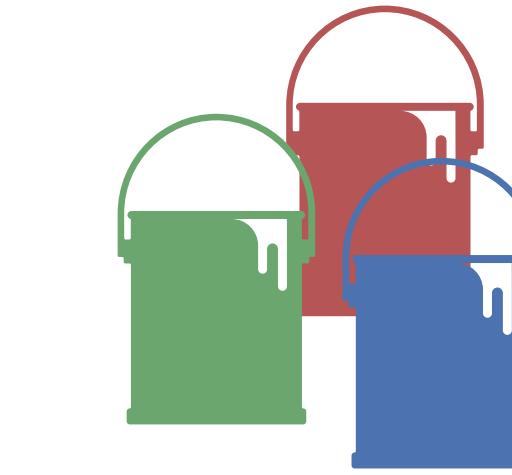
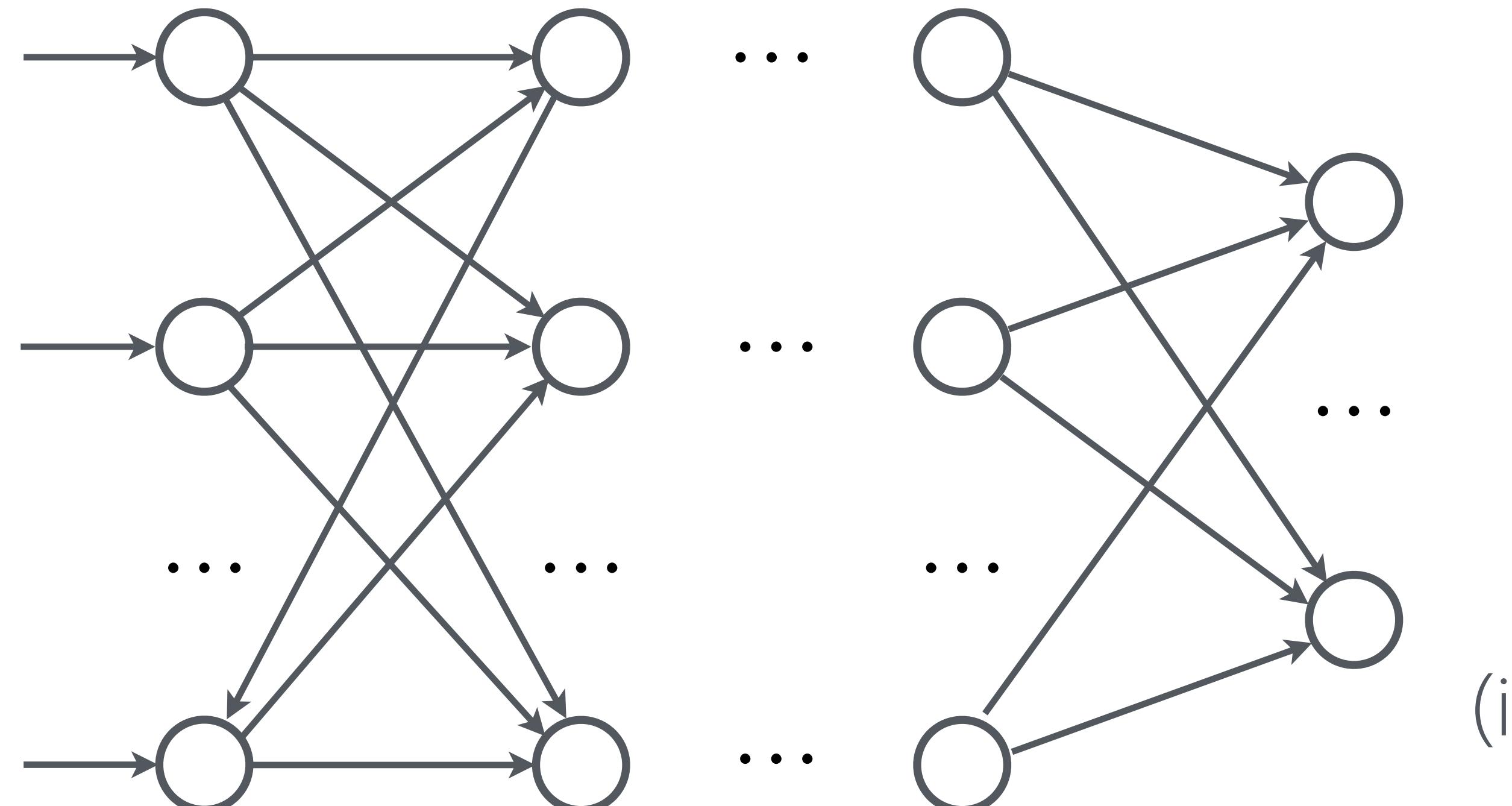


Fair = no discrimination

# Neural Network Analysis

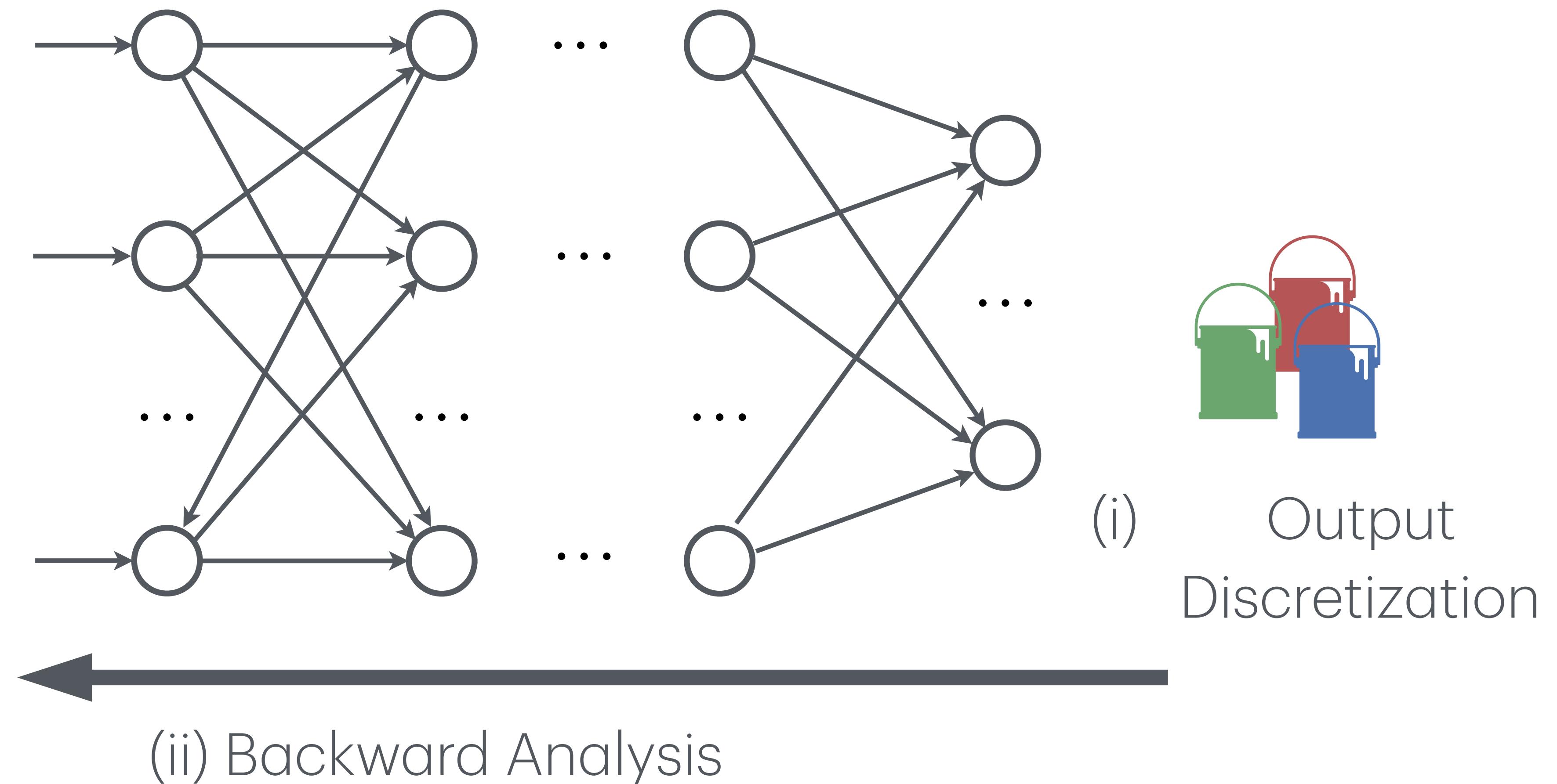


# Neural Network Analysis

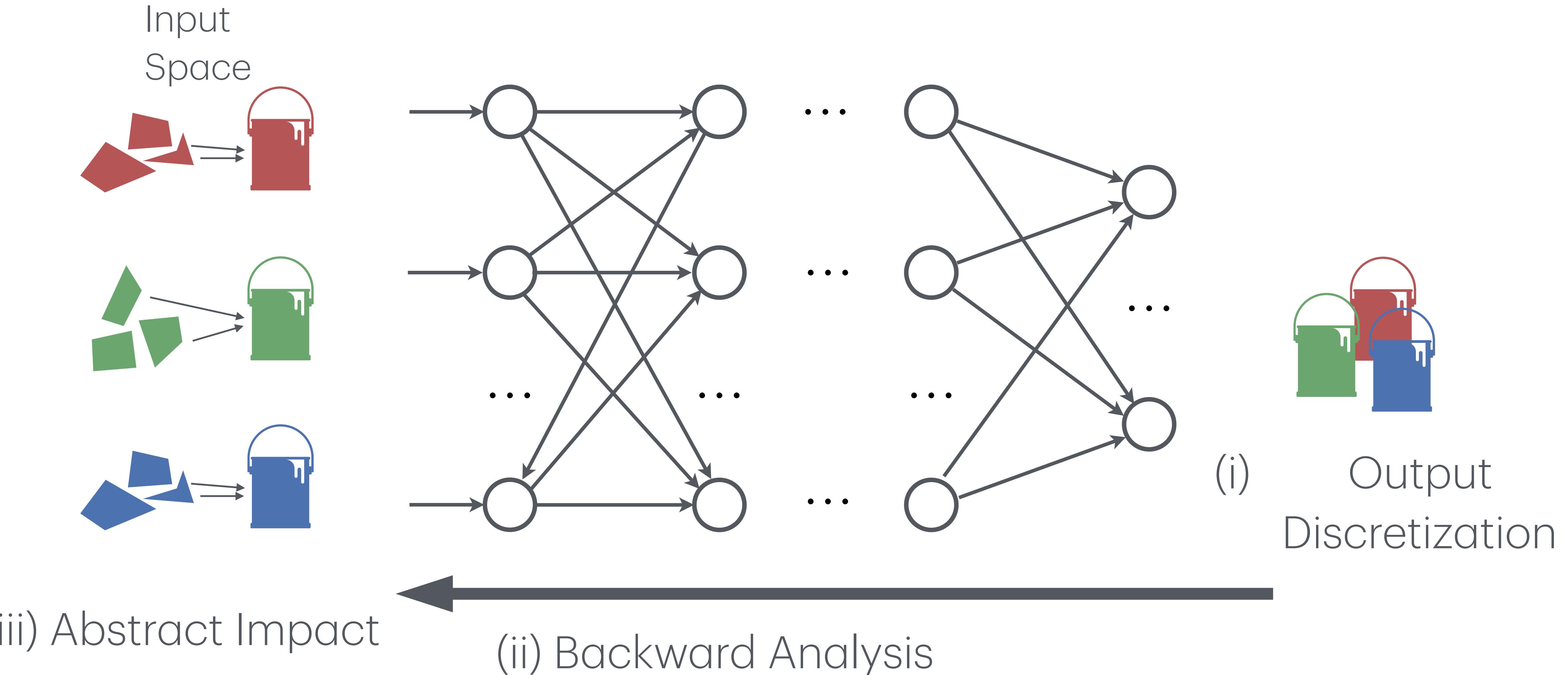


(i) Output  
Discretization

# Neural Network Analysis

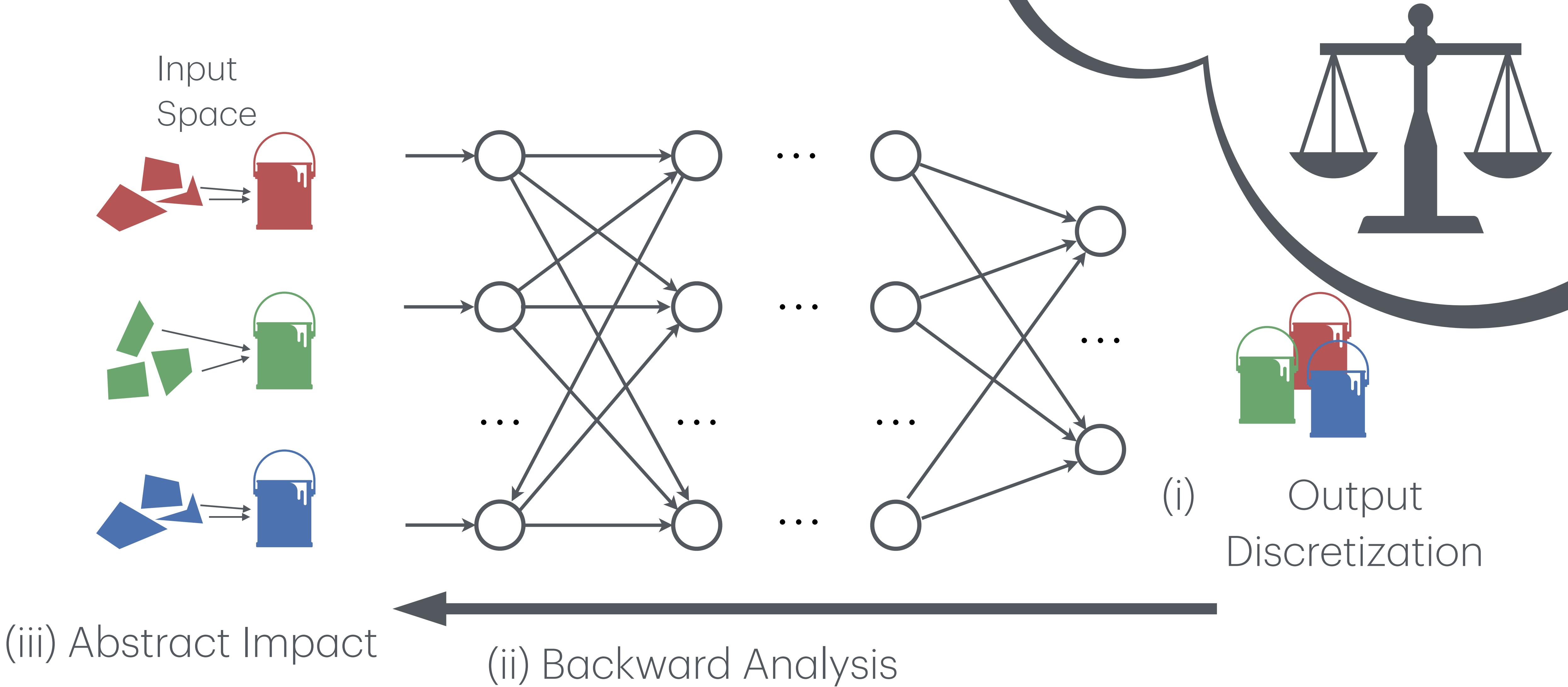


# Neural Network Analysis

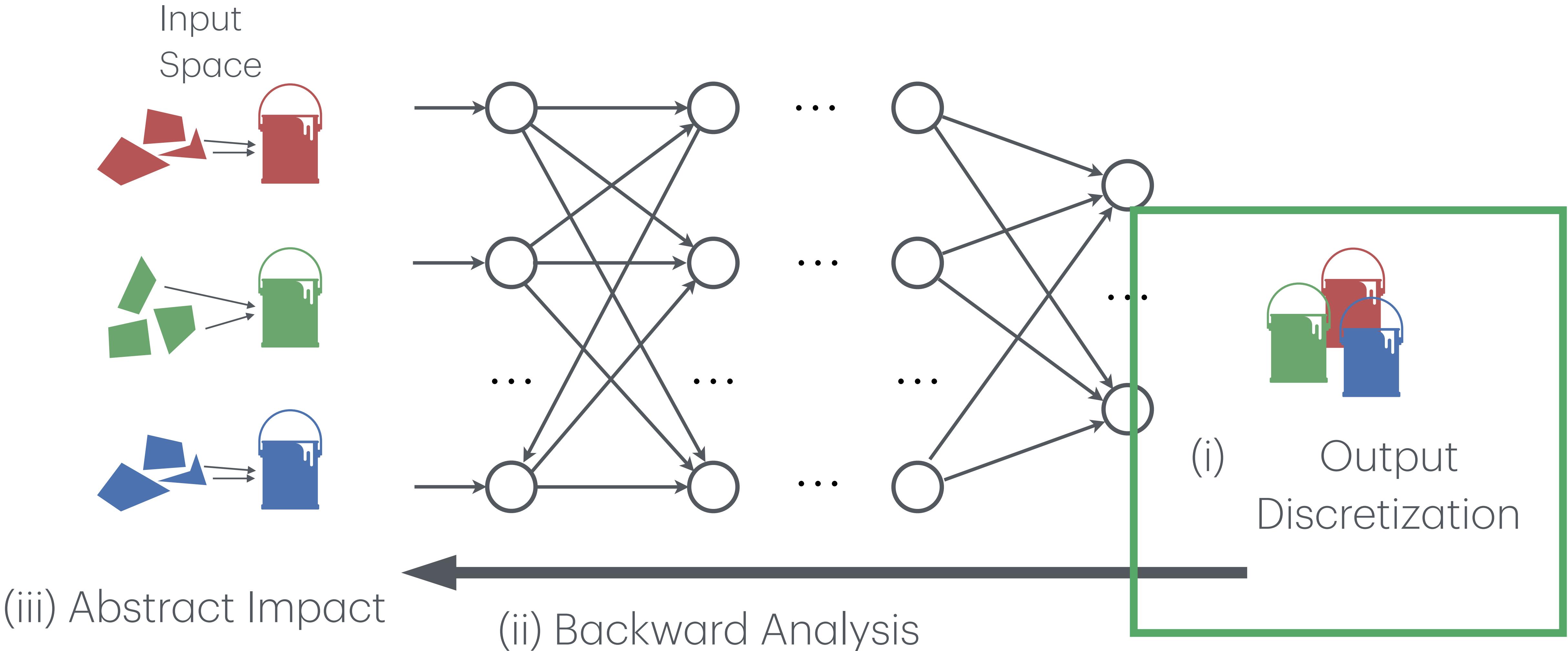


# Neural Network Analysis

Libra tool for  
qualitative fairness

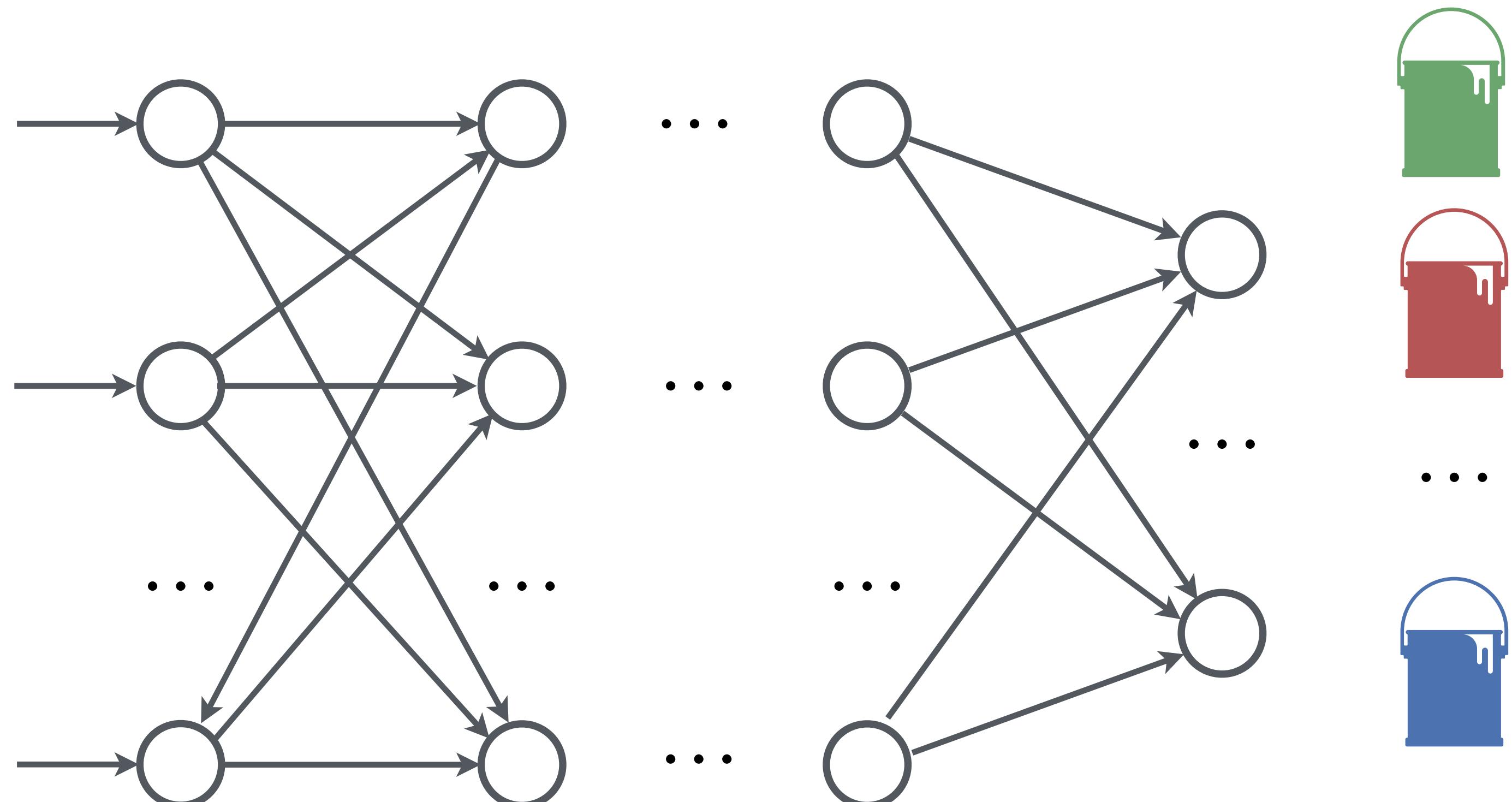


# Neural Network Analysis

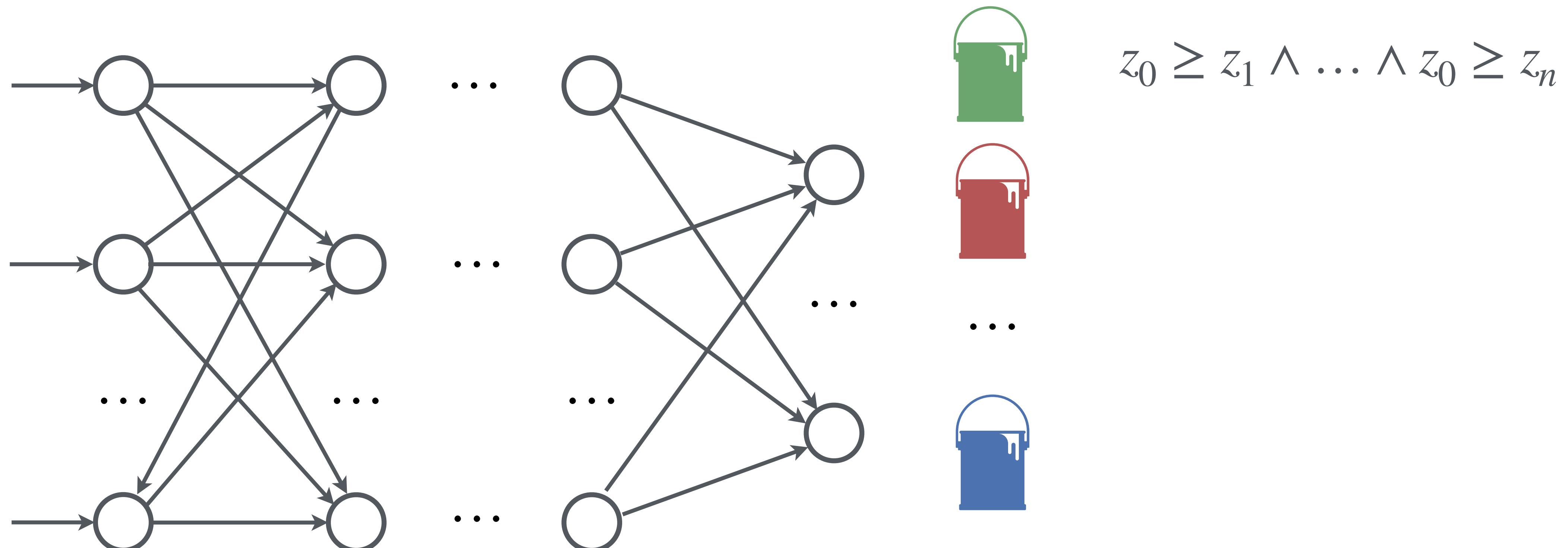


# (i) Output Discretization

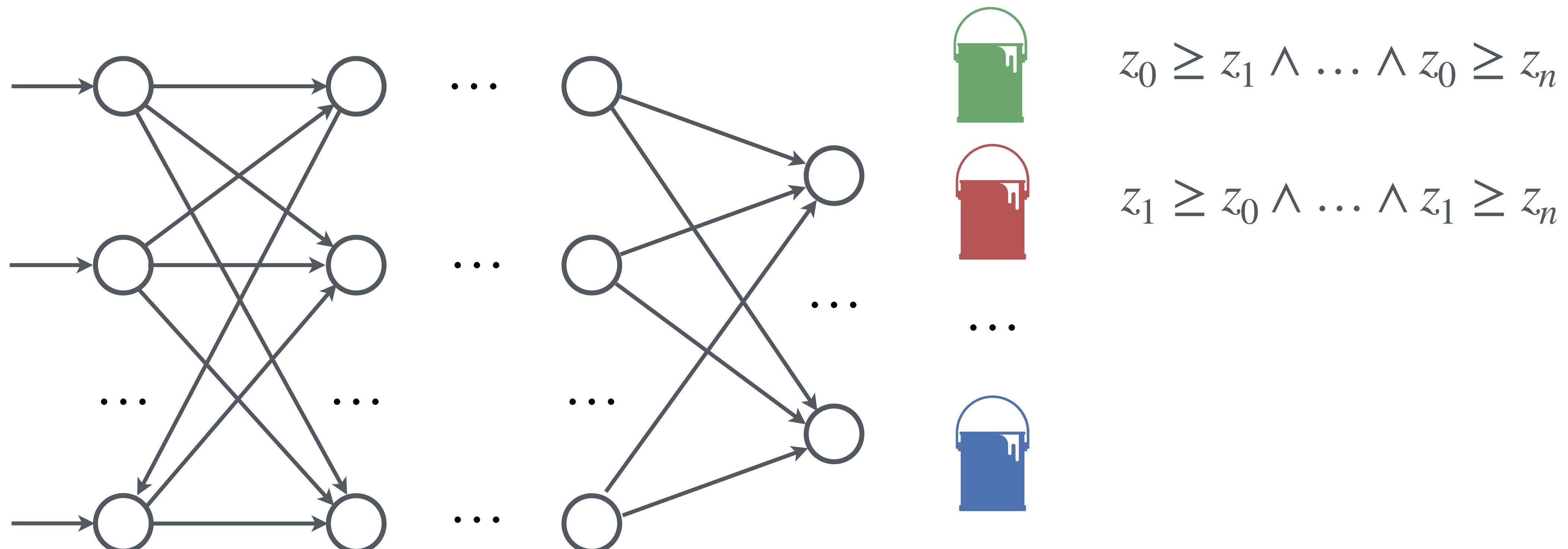
---



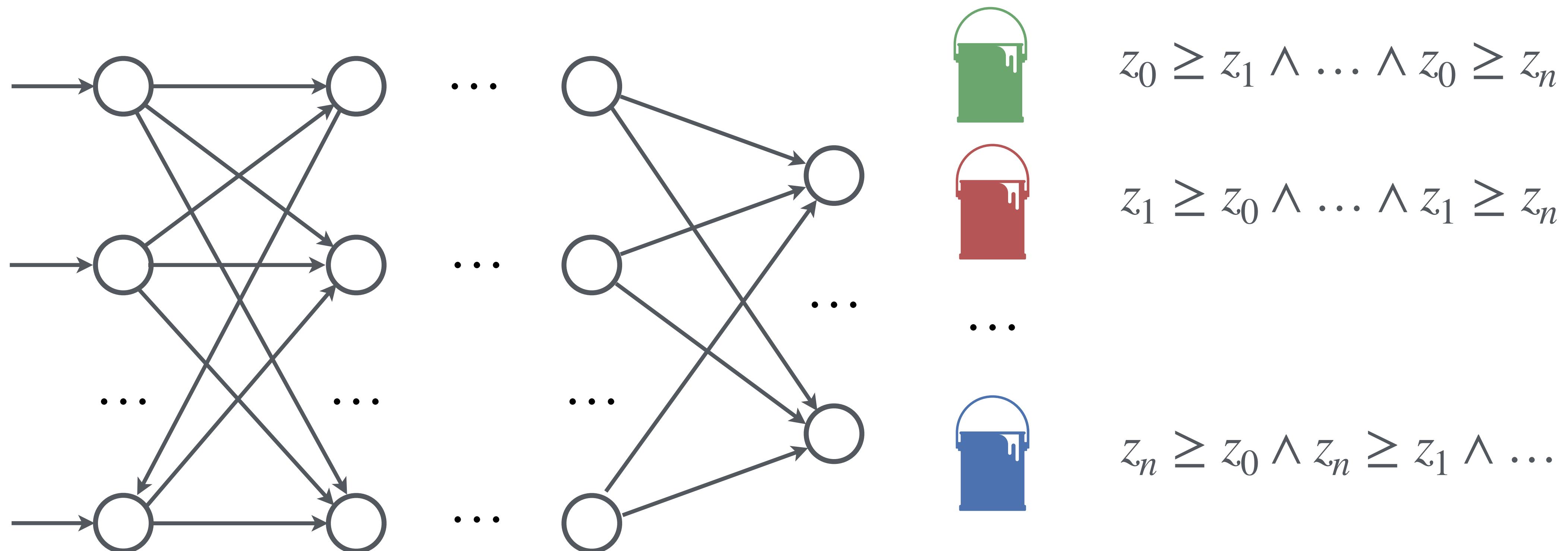
# (i) Output Discretization



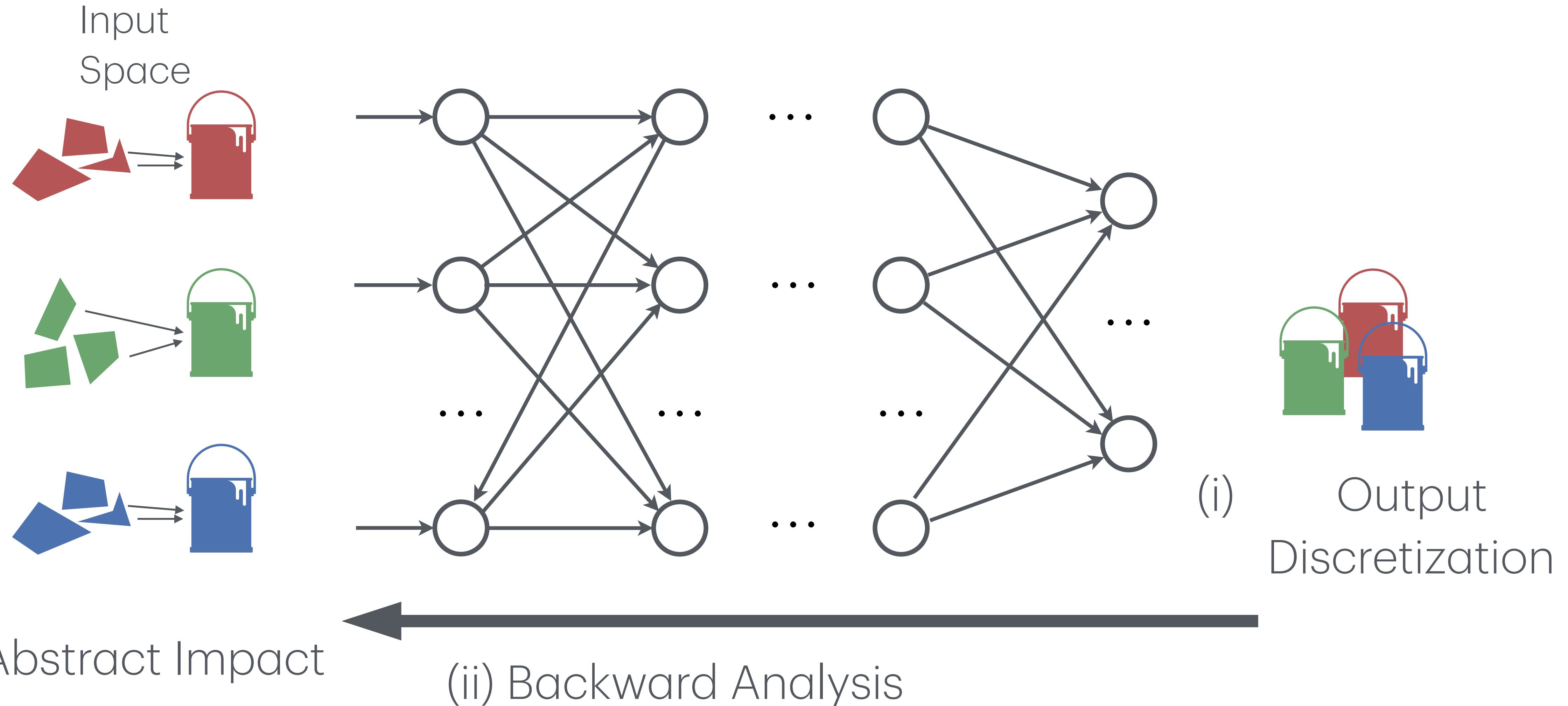
# (i) Output Discretization



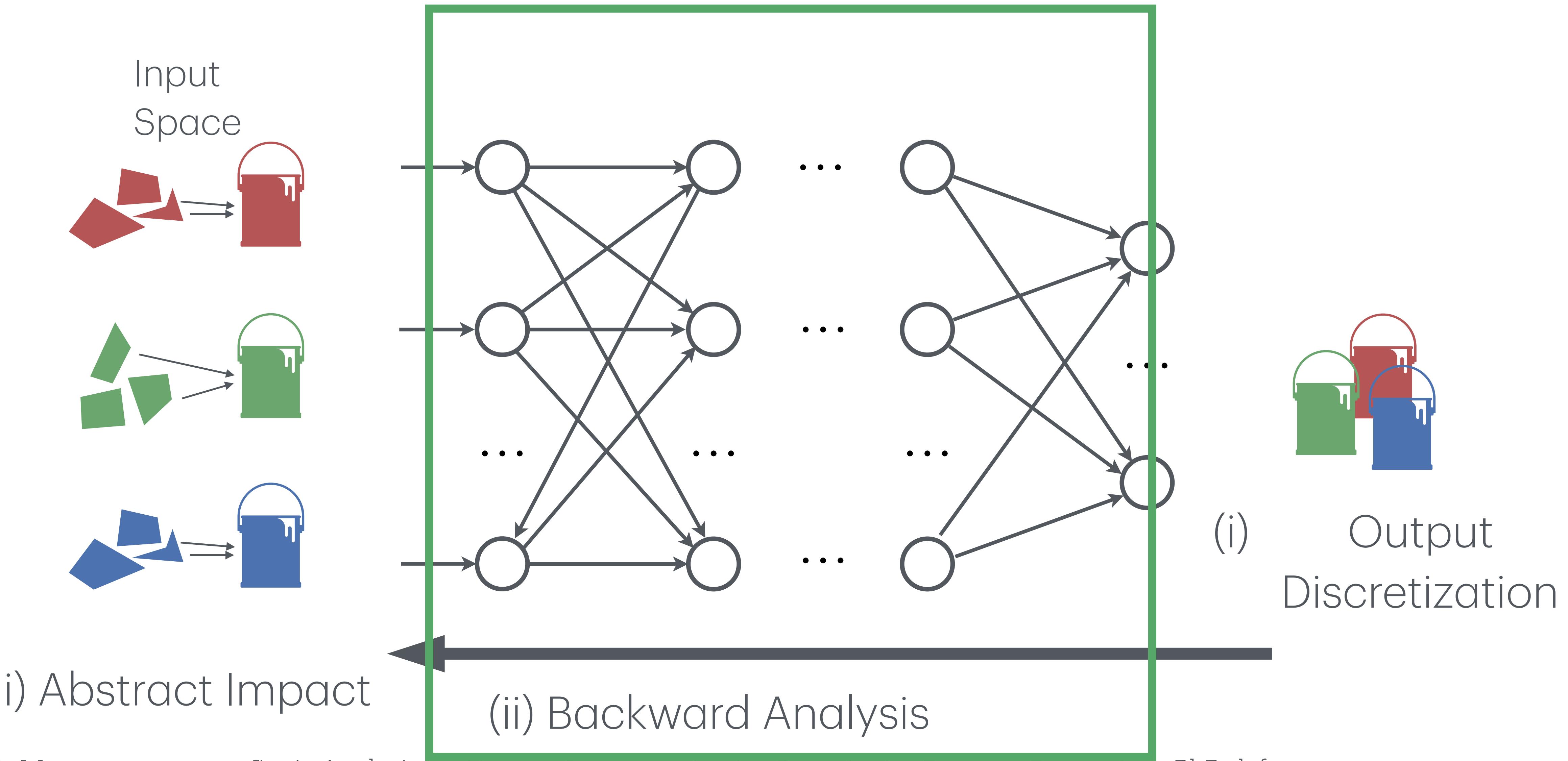
# (i) Output Discretization



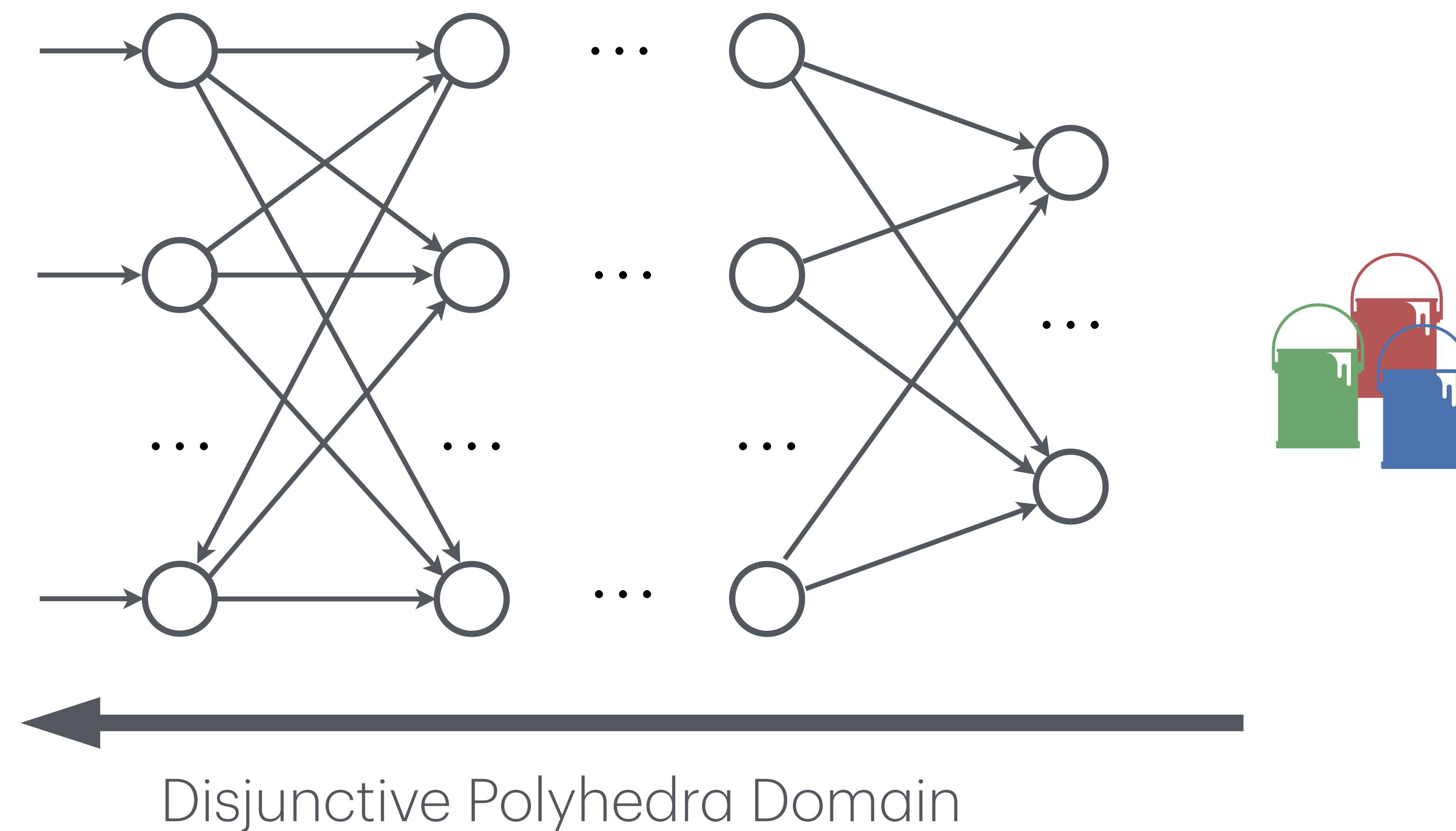
# Neural Network Analysis



# Neural Network Analysis

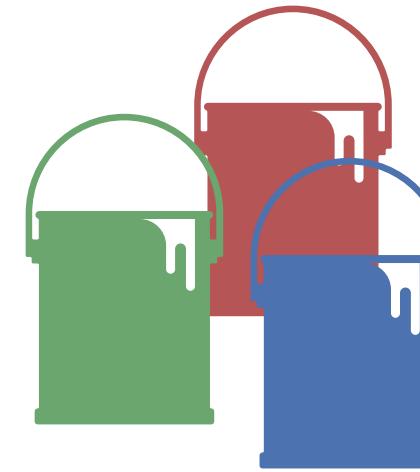
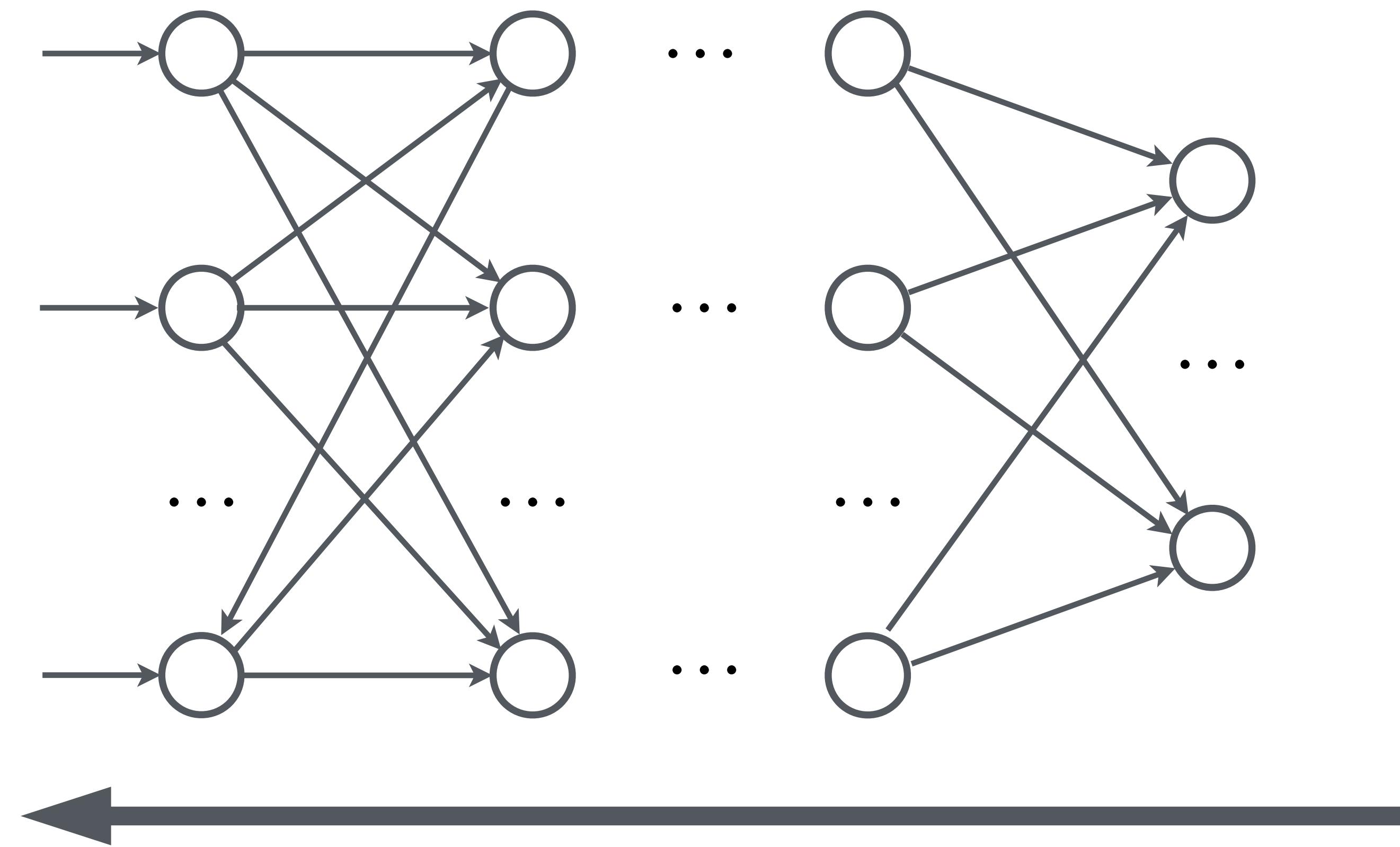


## (ii) Backward Abstract Analysis



## (ii) Backward Abstract Analysis

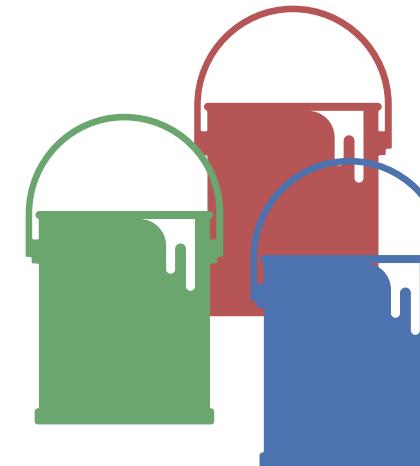
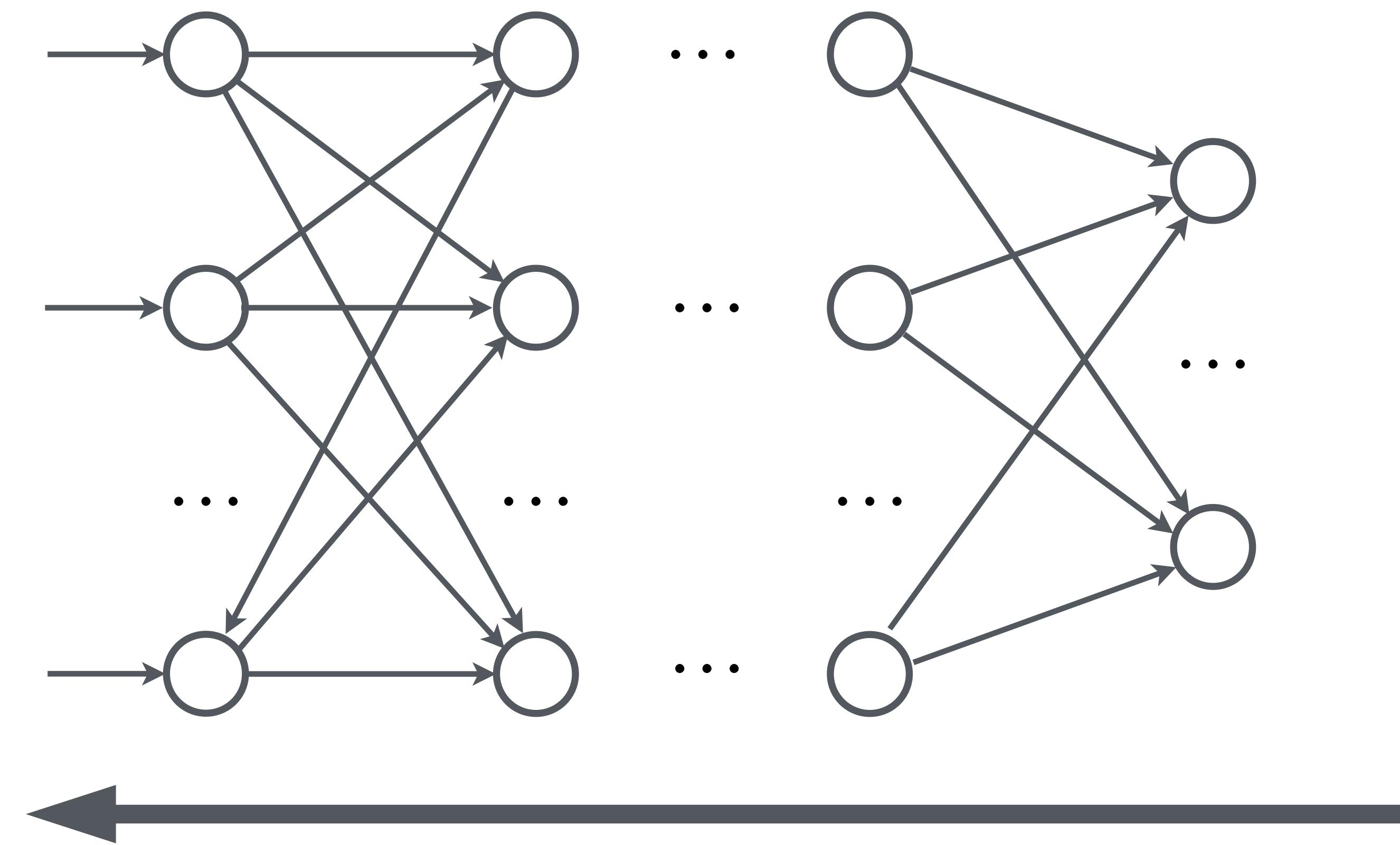
Highly precise



# (ii) Backward Abstract Analysis

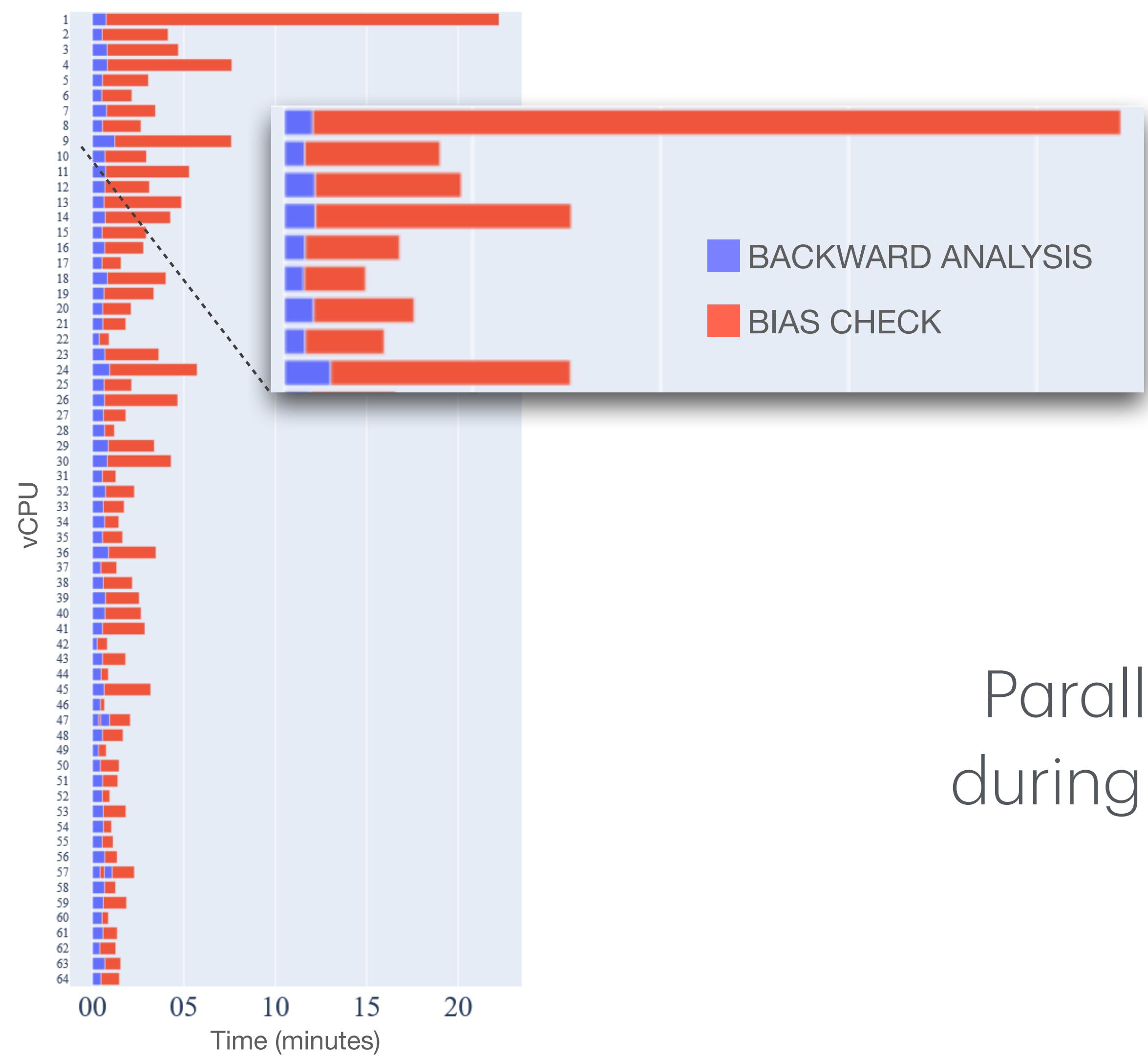
Highly precise

Quite slow



# Perfectly Parallelization

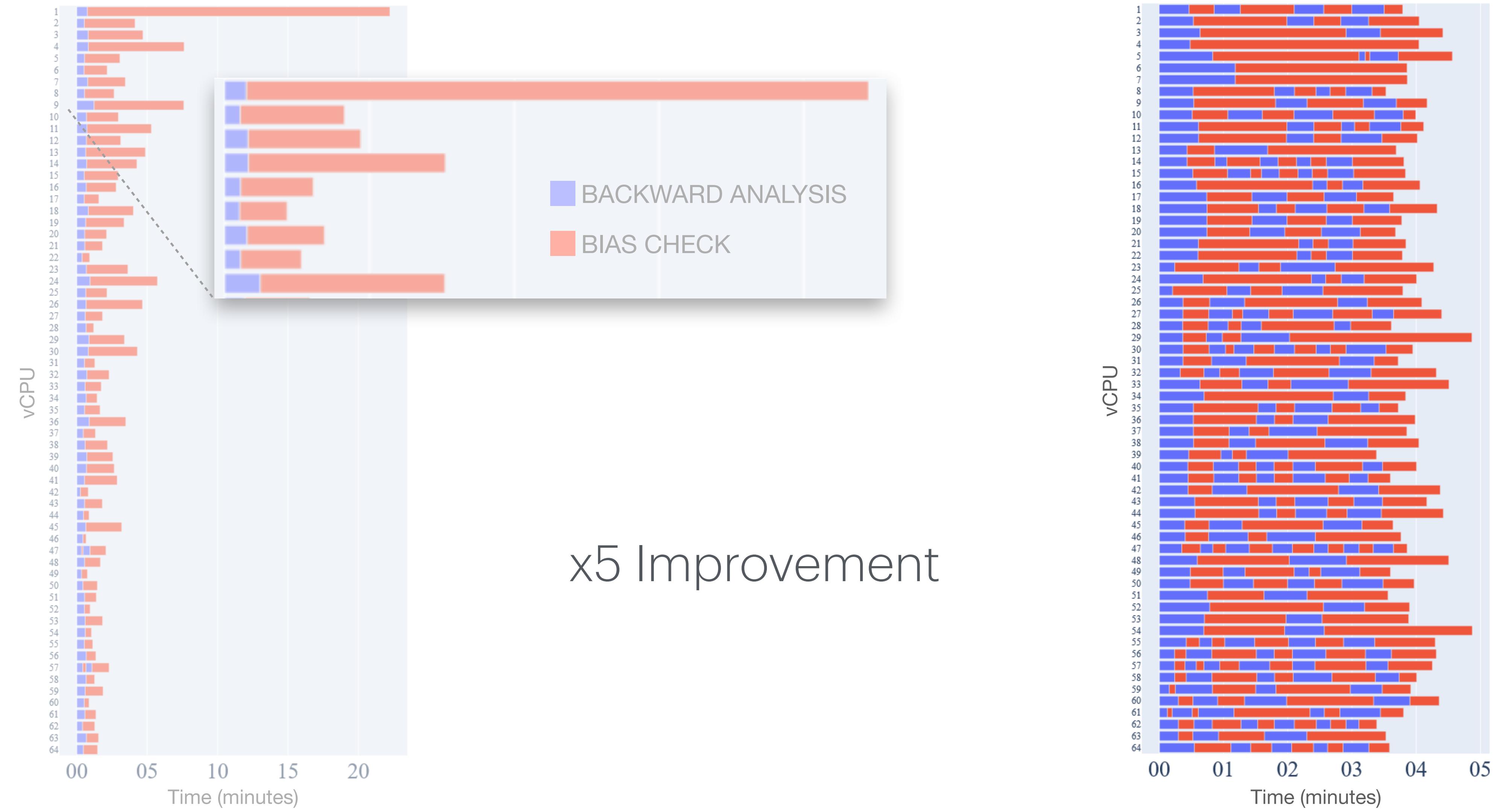
Libra tool for qualitative fairness



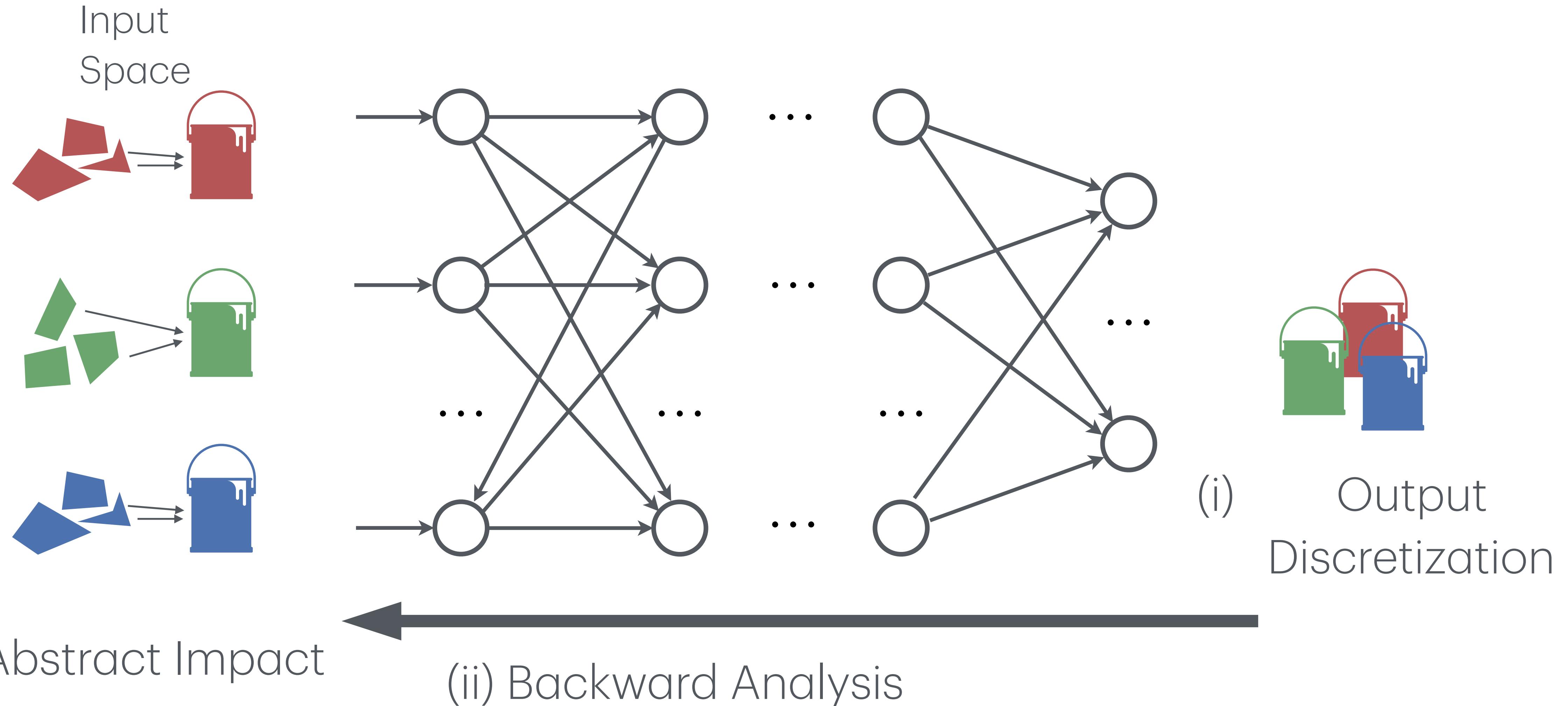
Parallelization of each path  
during the backward analysis



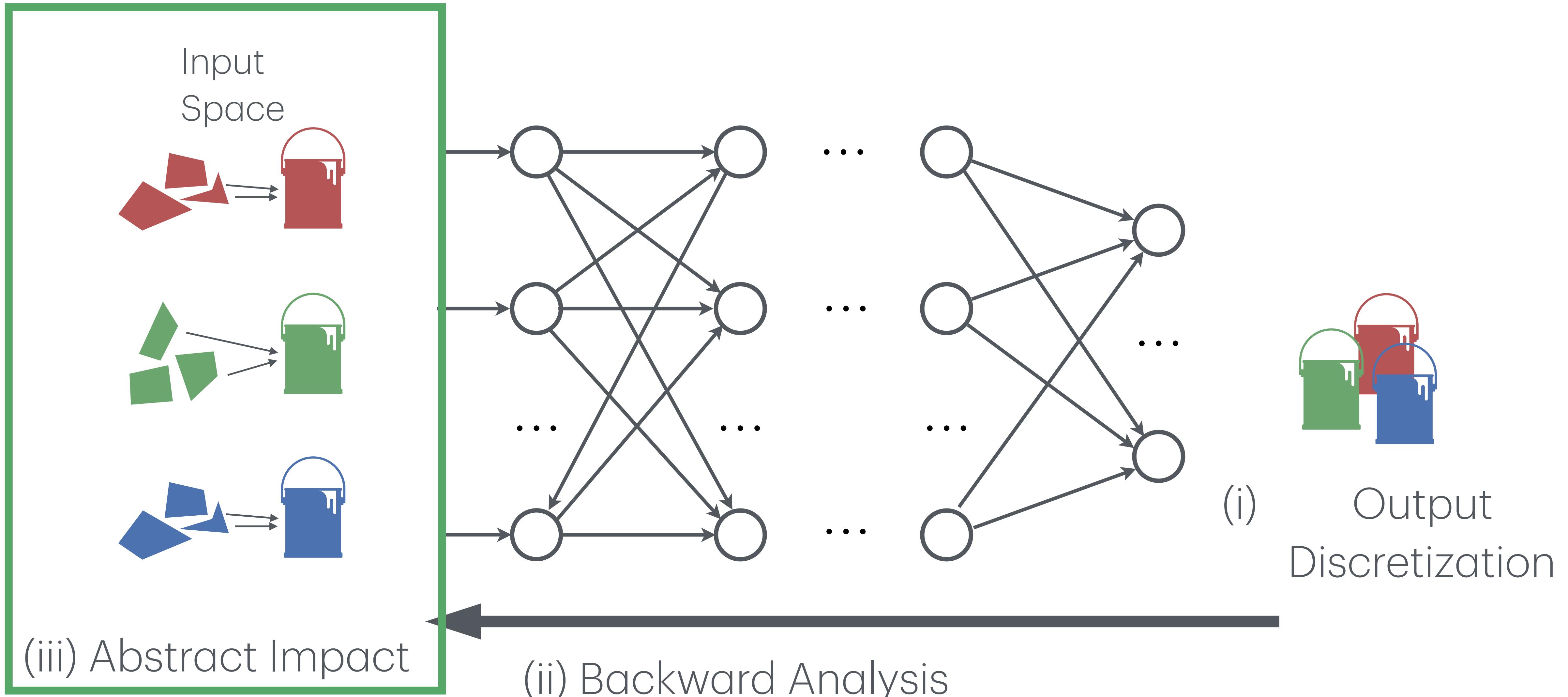
# Perfectly Parallelization



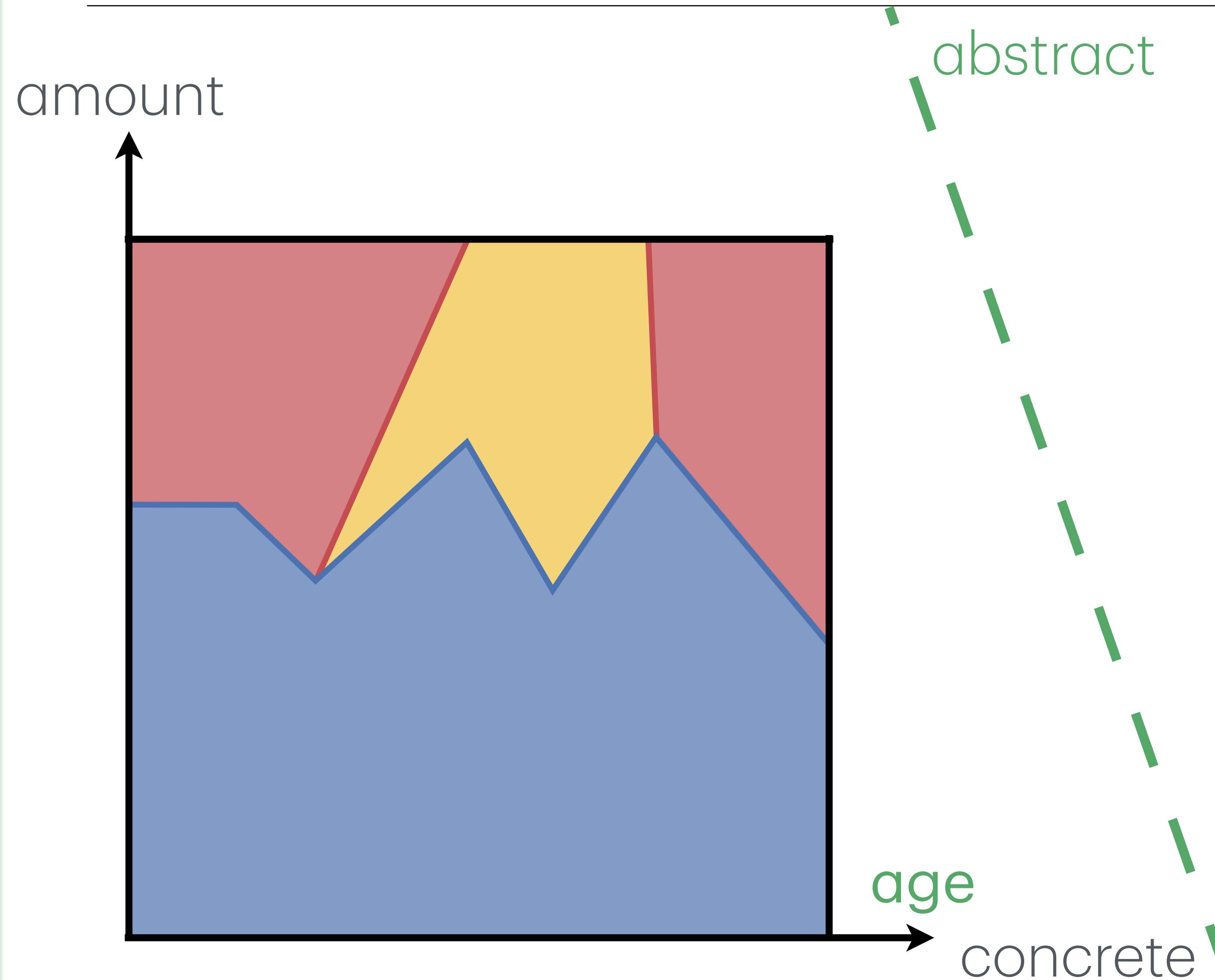
# Neural Network Analysis



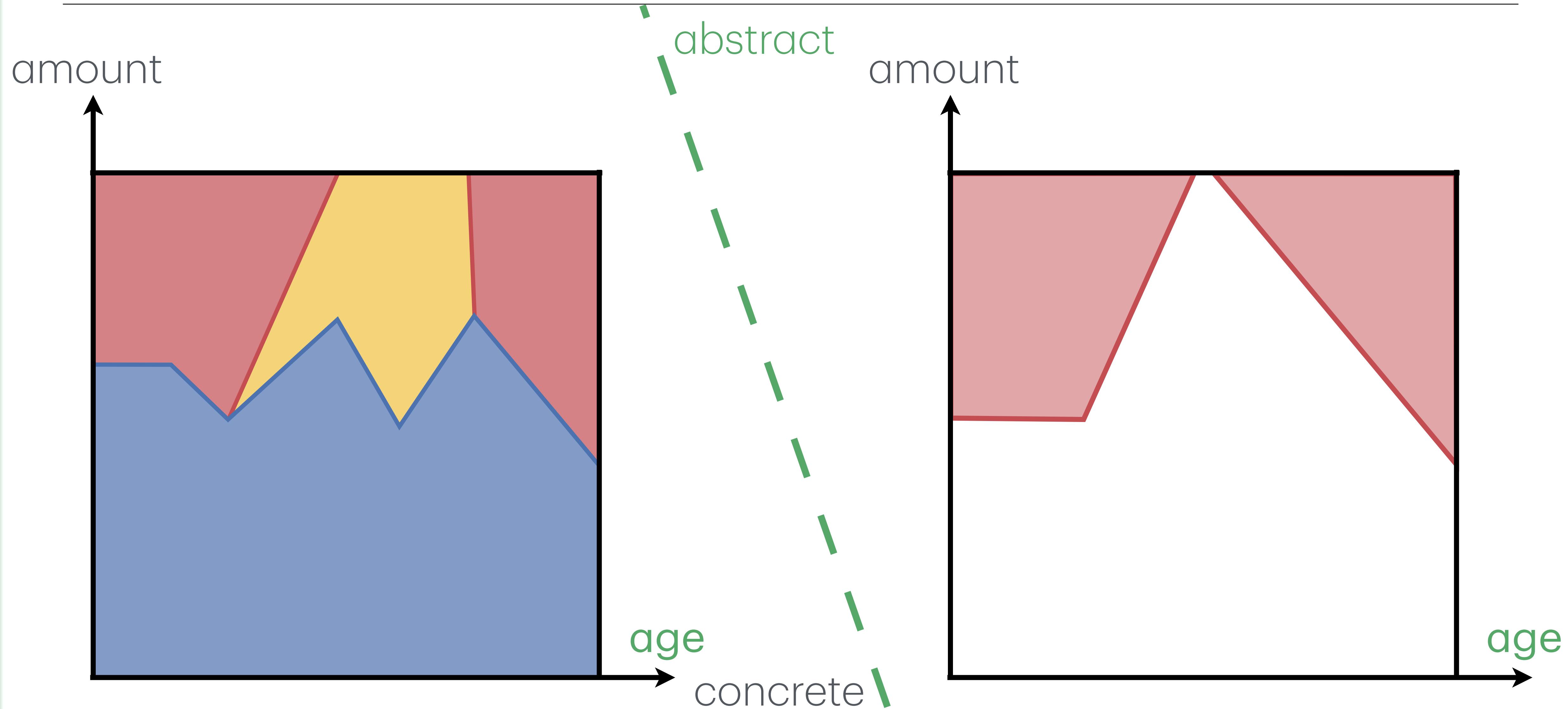
# Neural Network Analysis



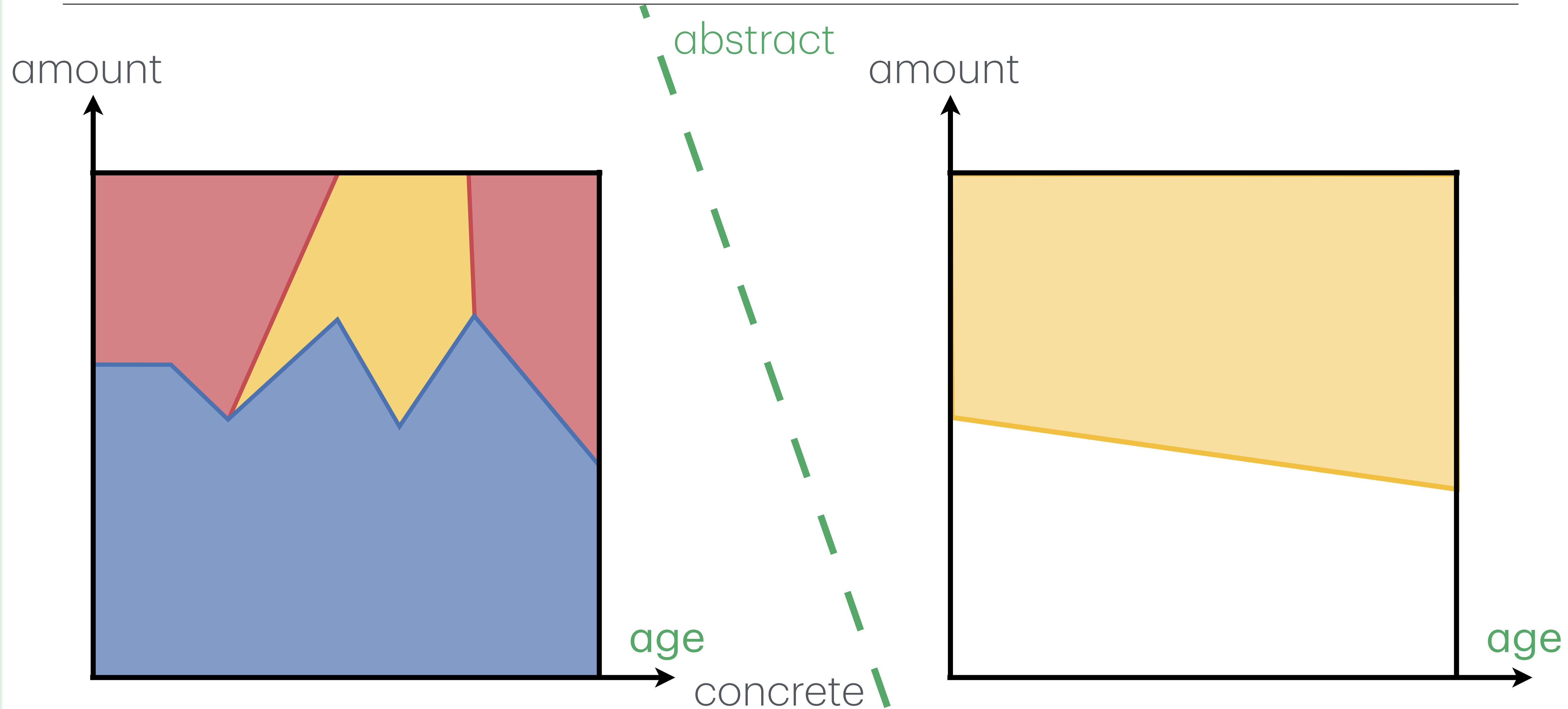
### (iii) Fair Input Space: QAUNUSED



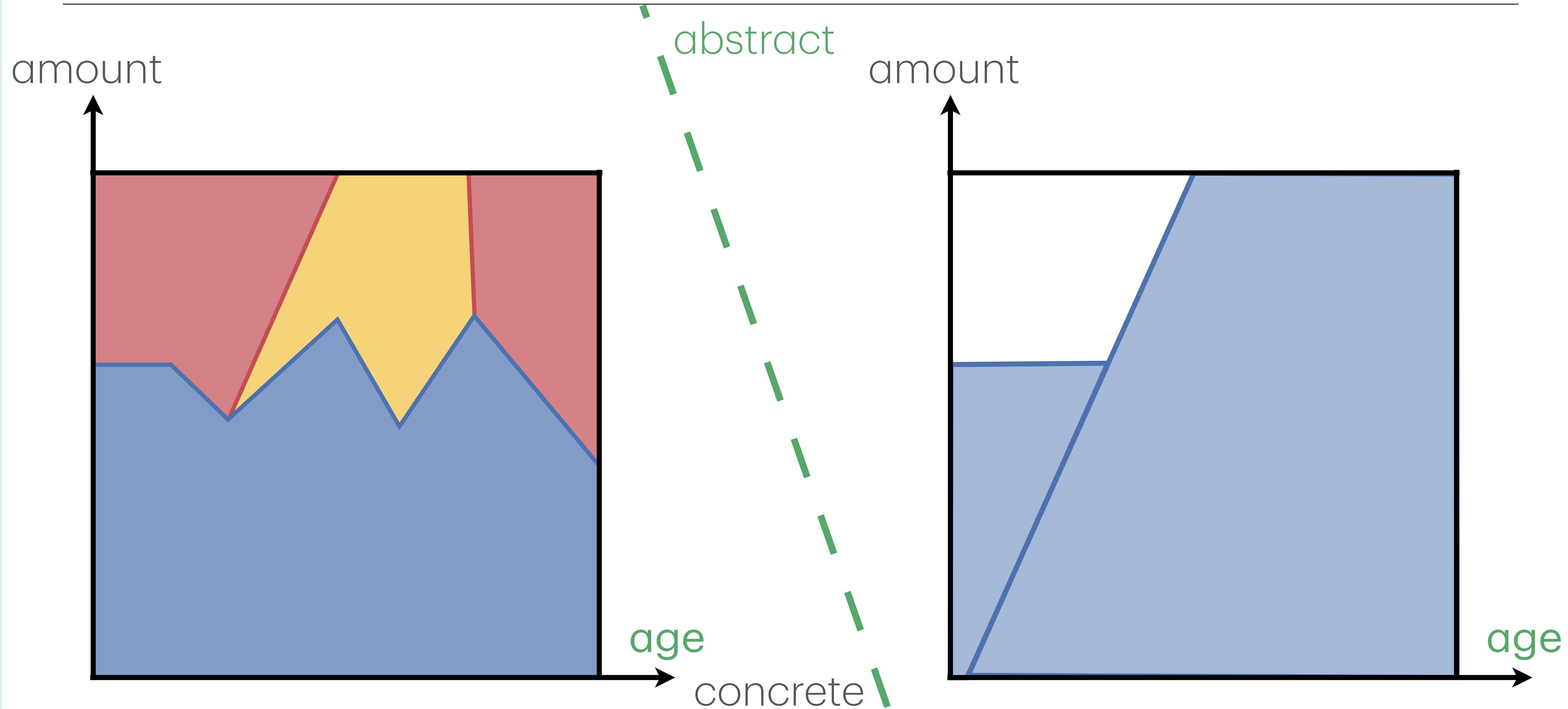
### (iii) Fair Input Space: QAUNUSED



### (iii) Fair Input Space: QAUNUSED

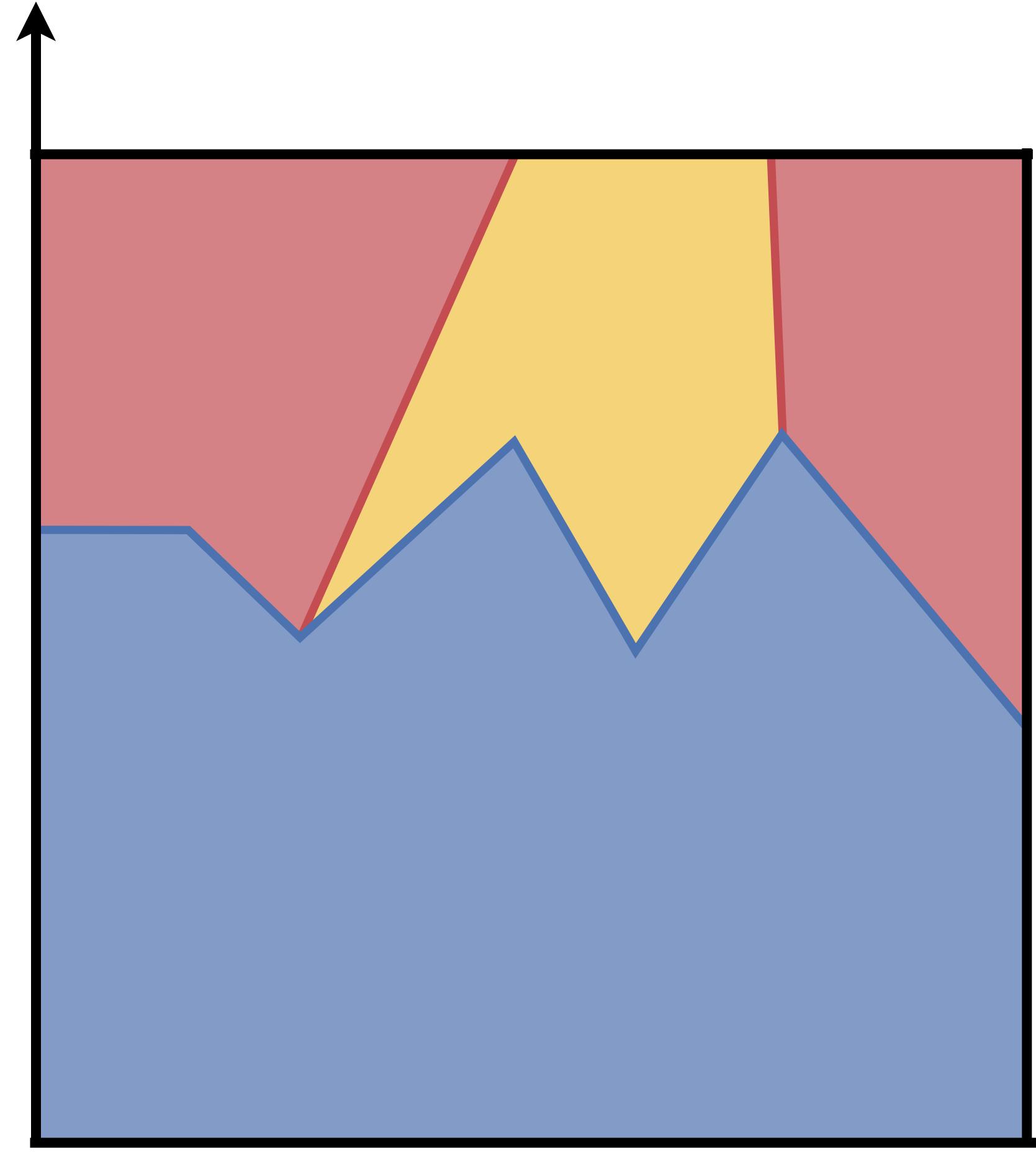


### (iii) Fair Input Space: QAUNUSED



### (iii) Fair Input Space: QAUNUSED

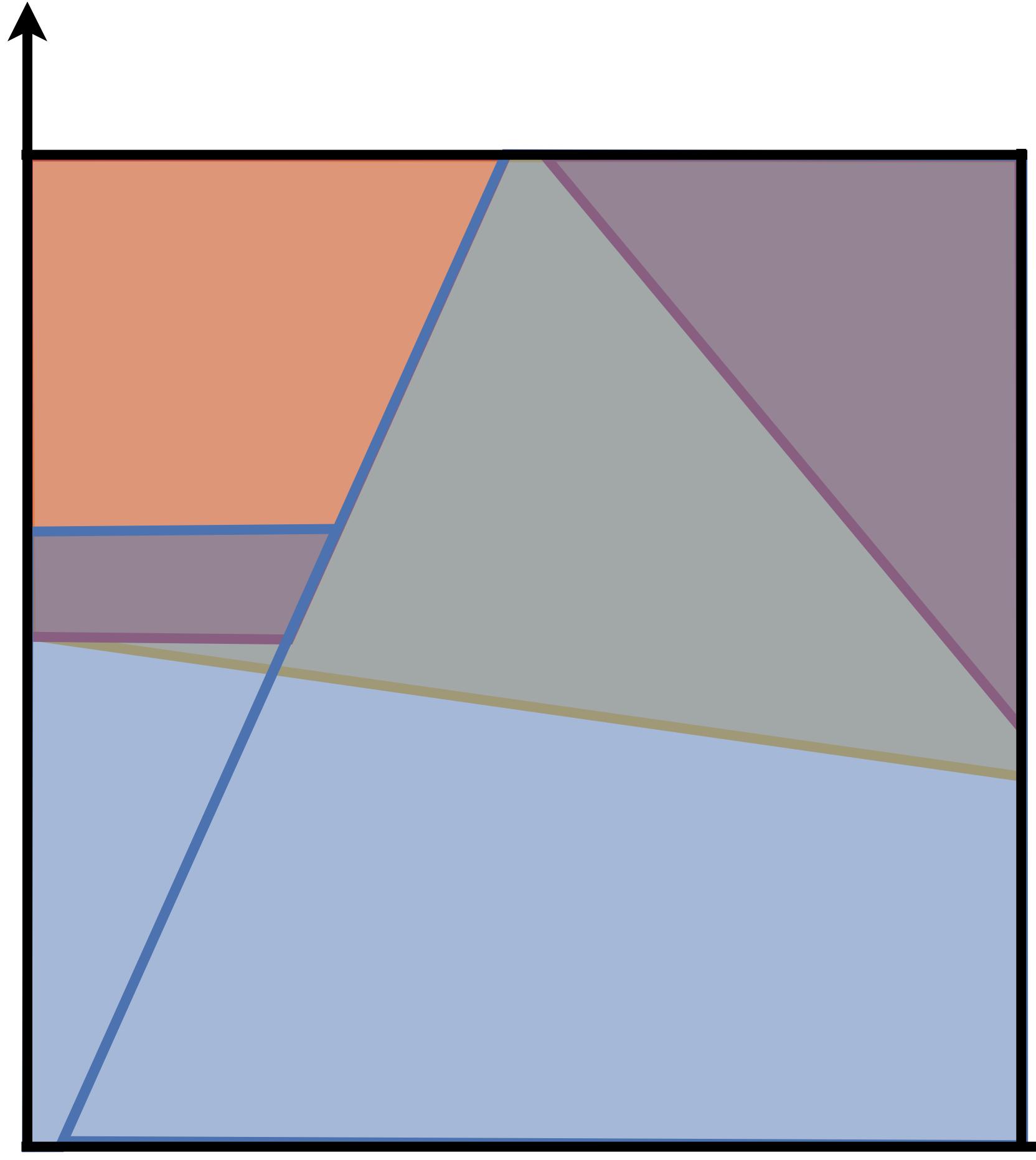
amount



age

abstract

amount

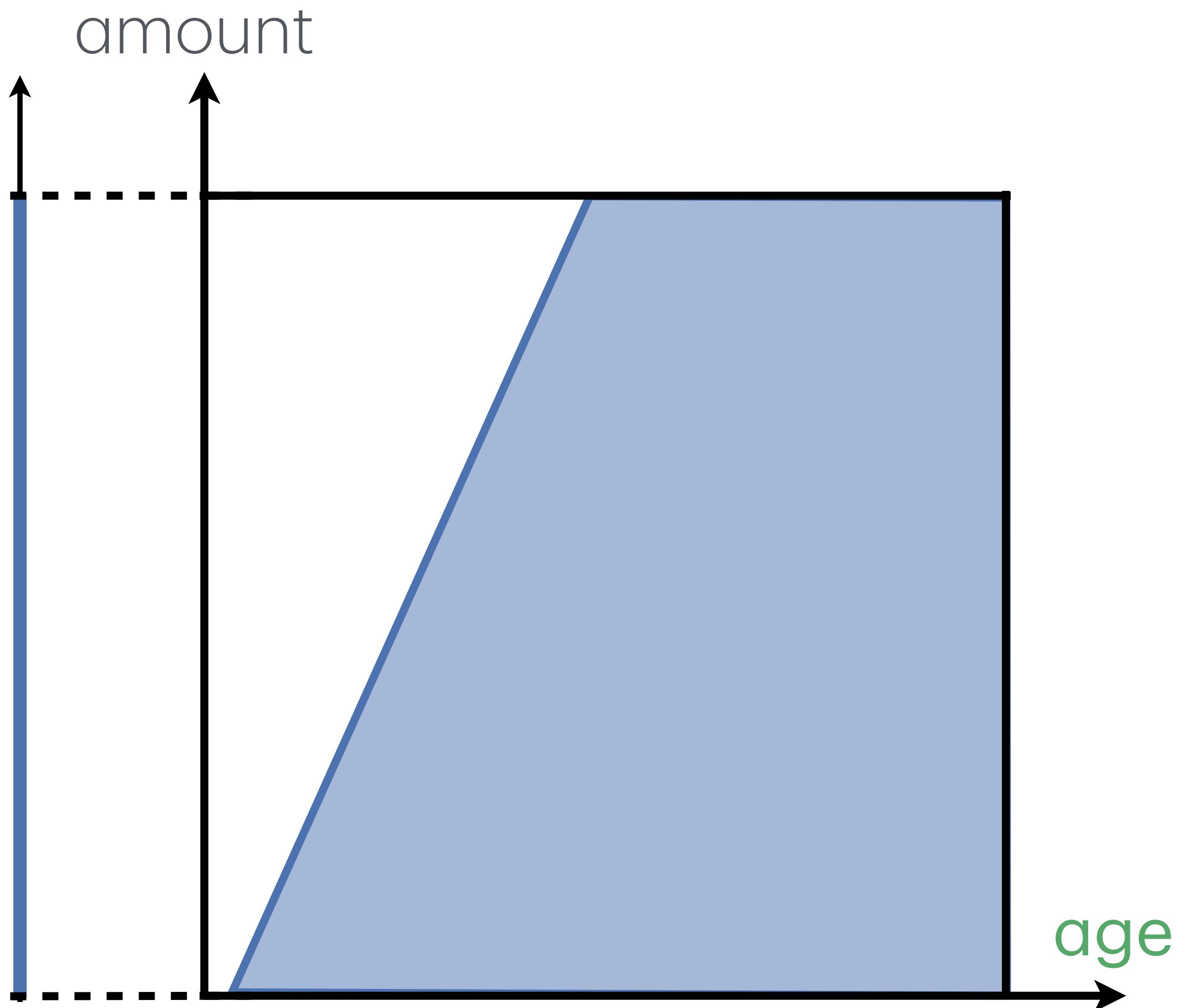


age

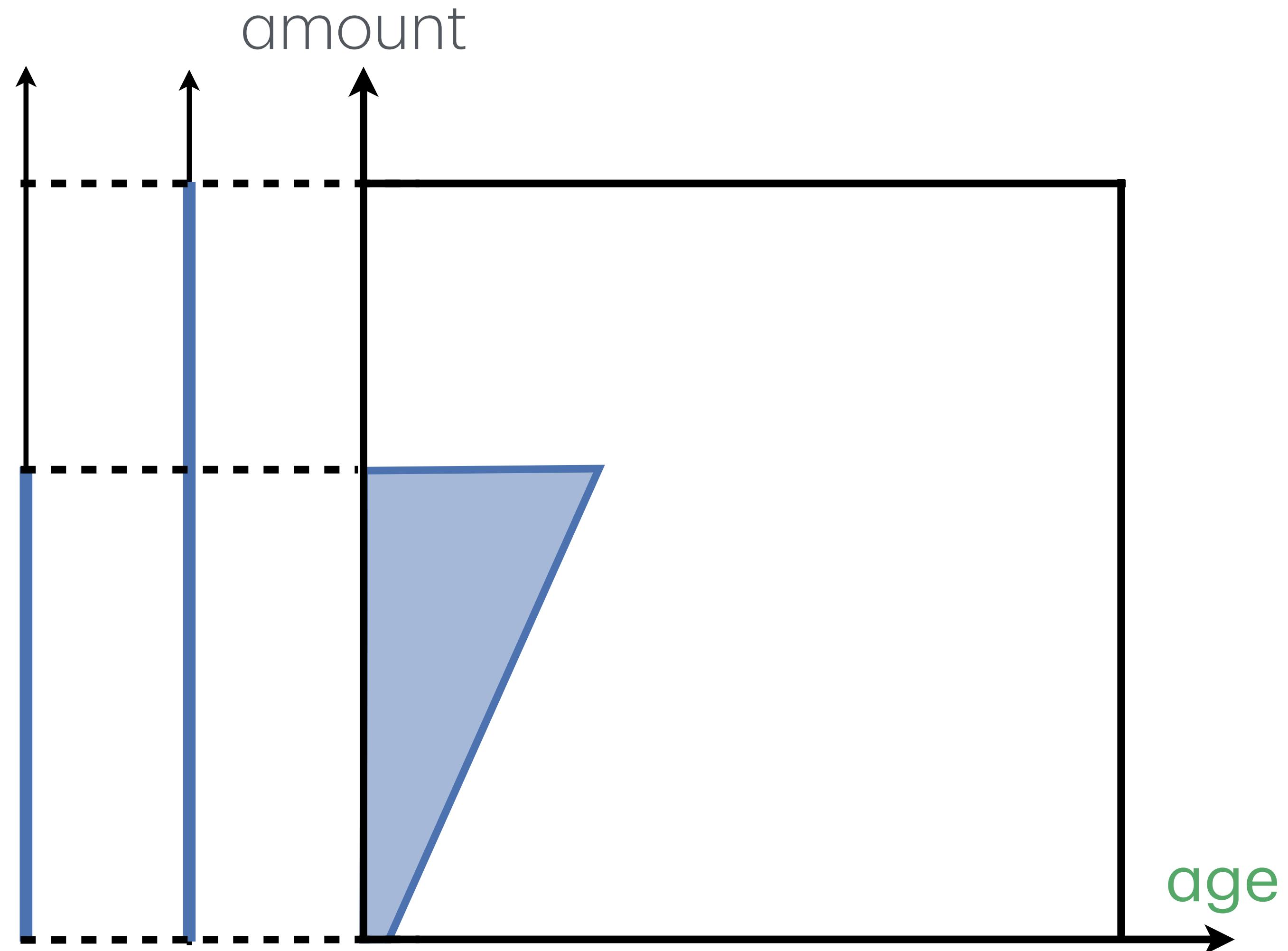
concrete

### (iii) Fair Input Space: QAUNUSED

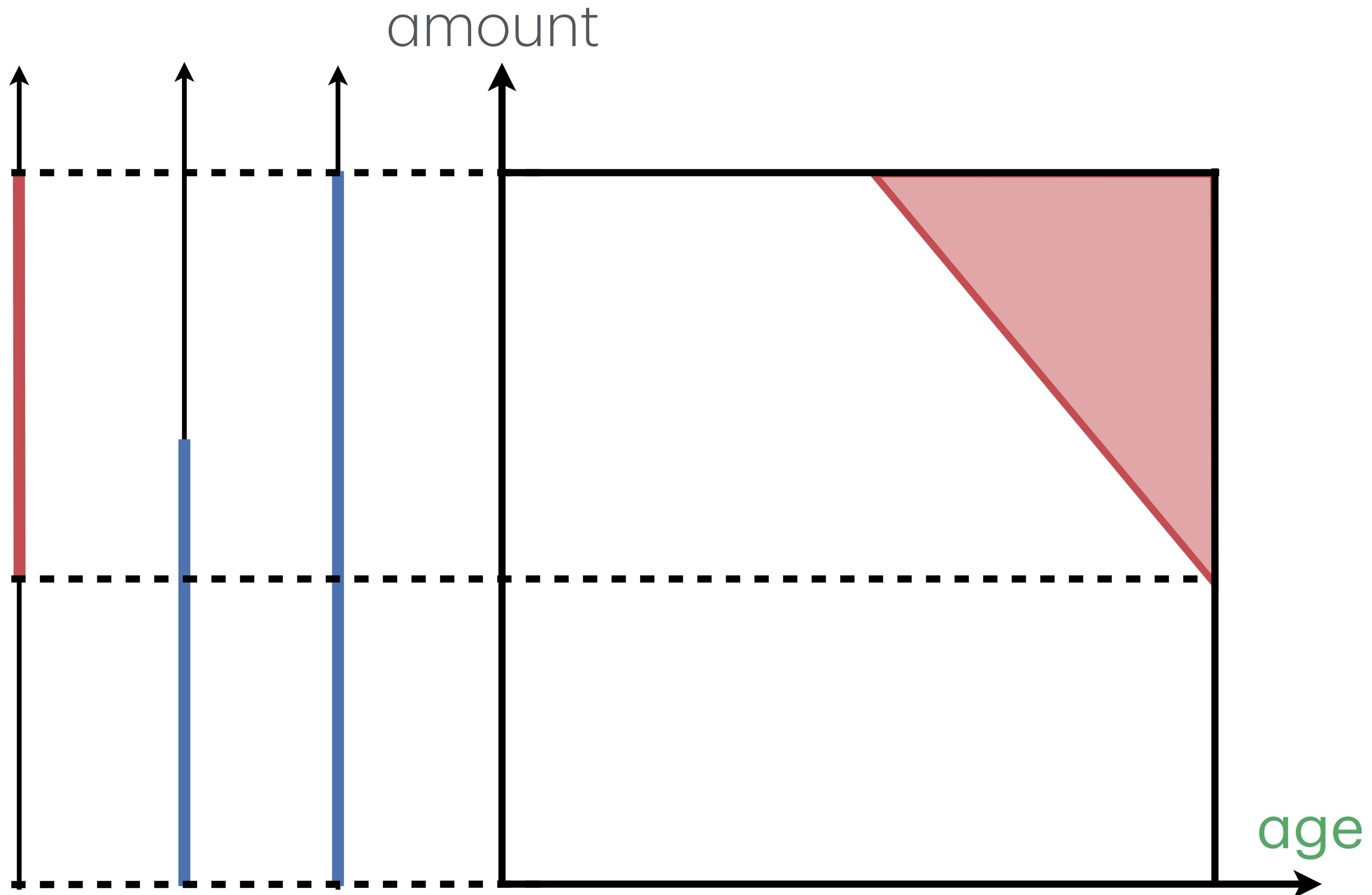
---



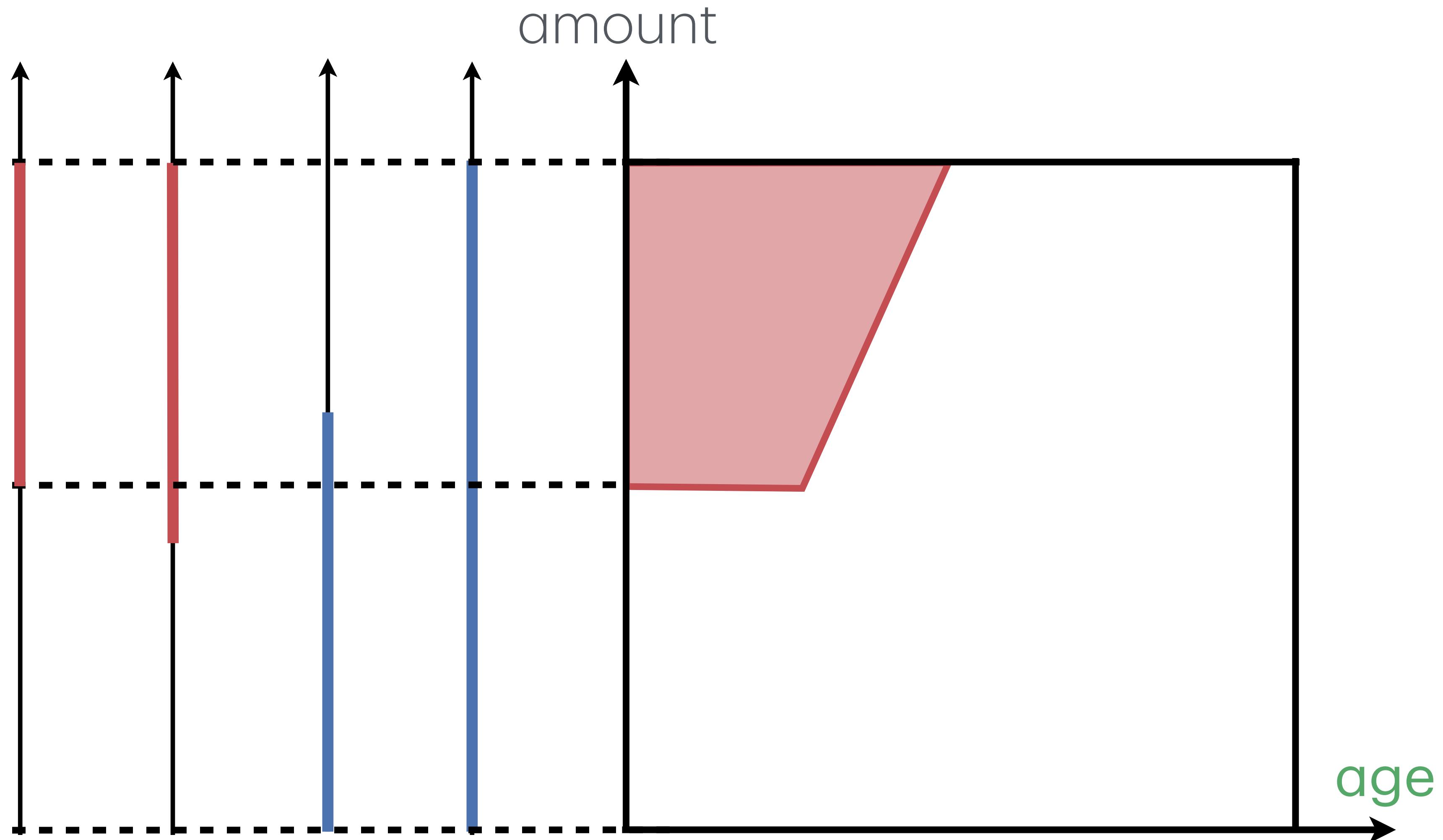
### (iii) Fair Input Space: QAUNUSED



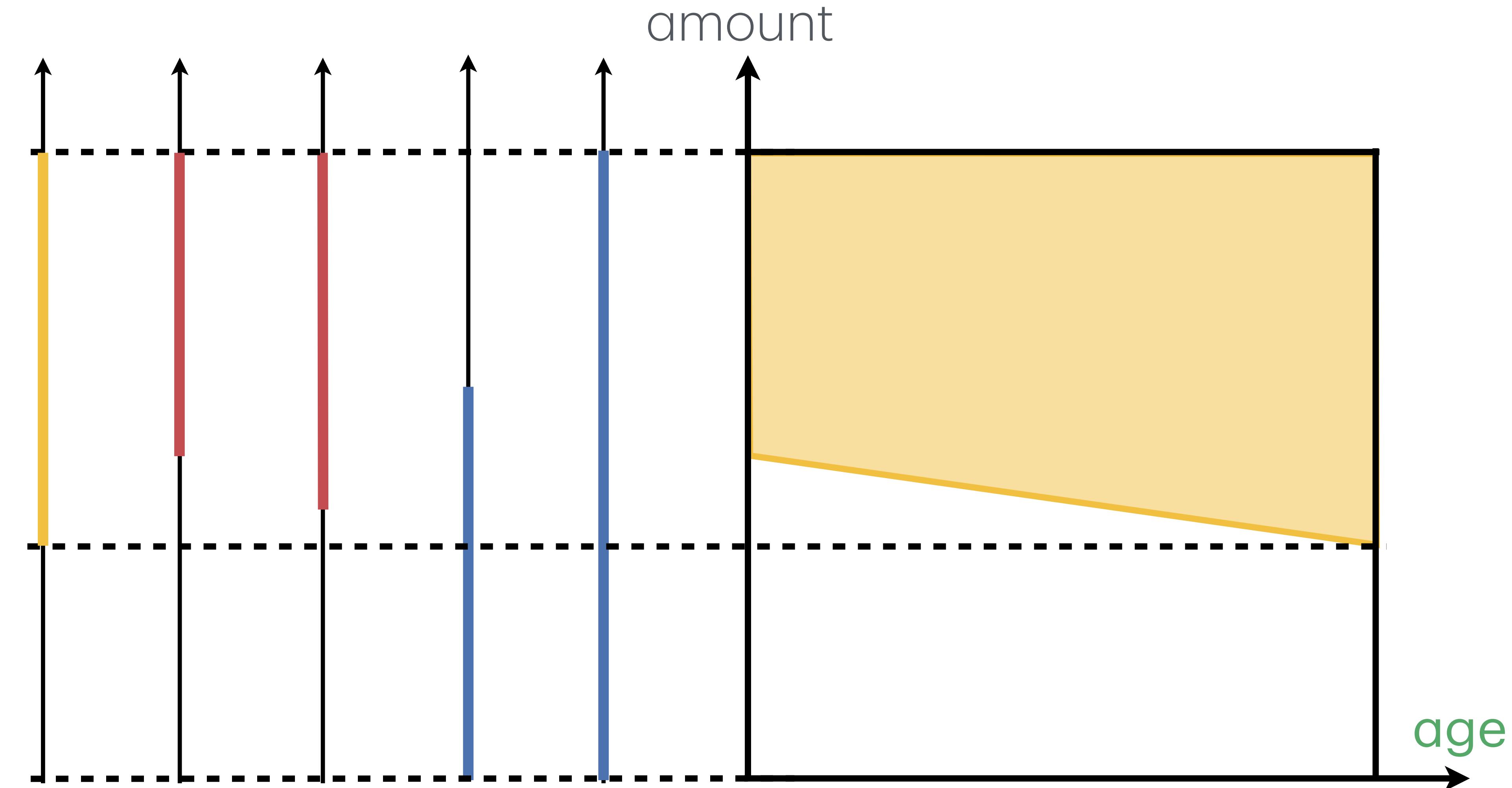
### (iii) Fair Input Space: QAUNUSED



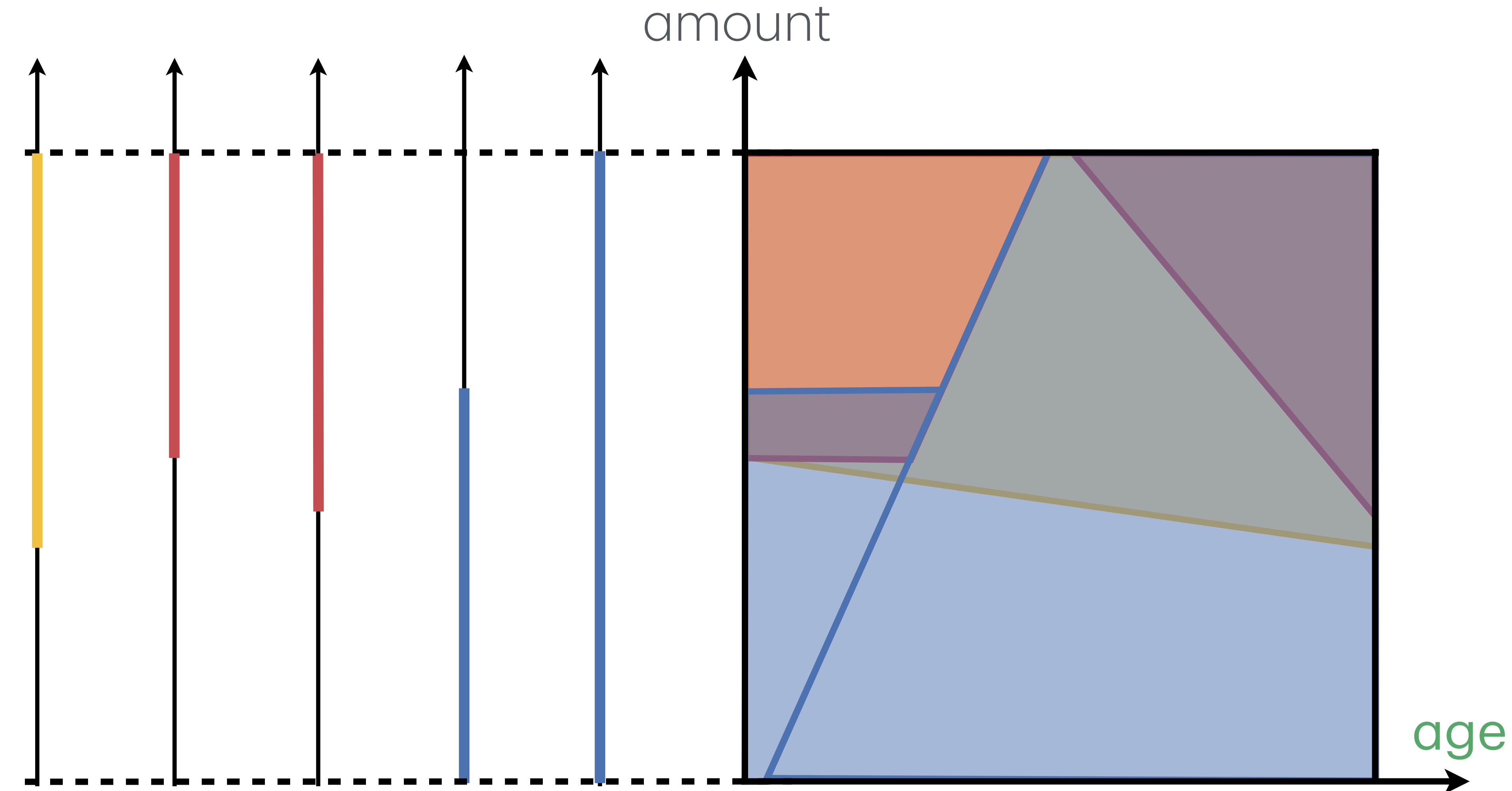
### (iii) Fair Input Space: QAUNUSED



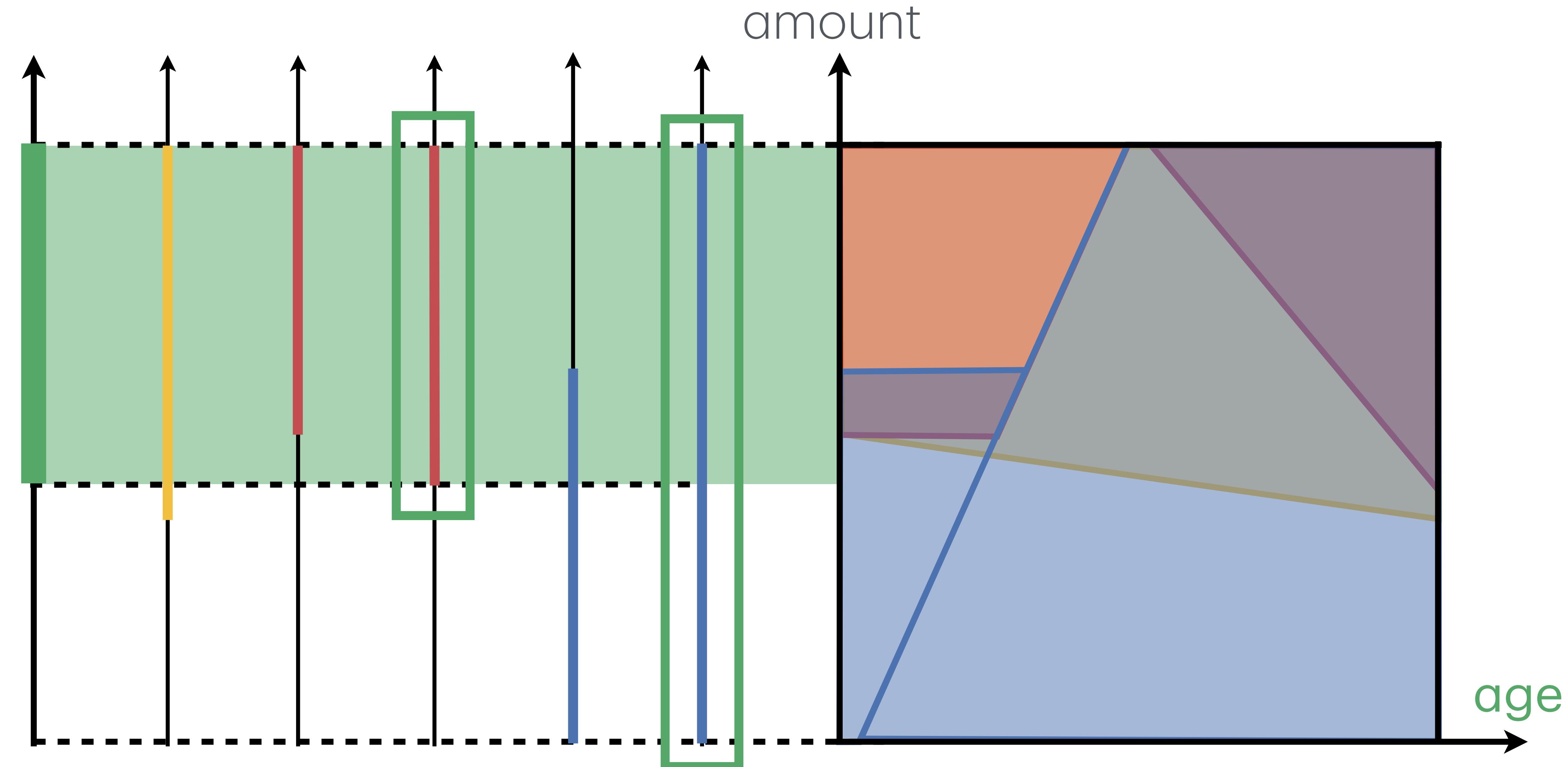
### (iii) Fair Input Space: QAUNUSED



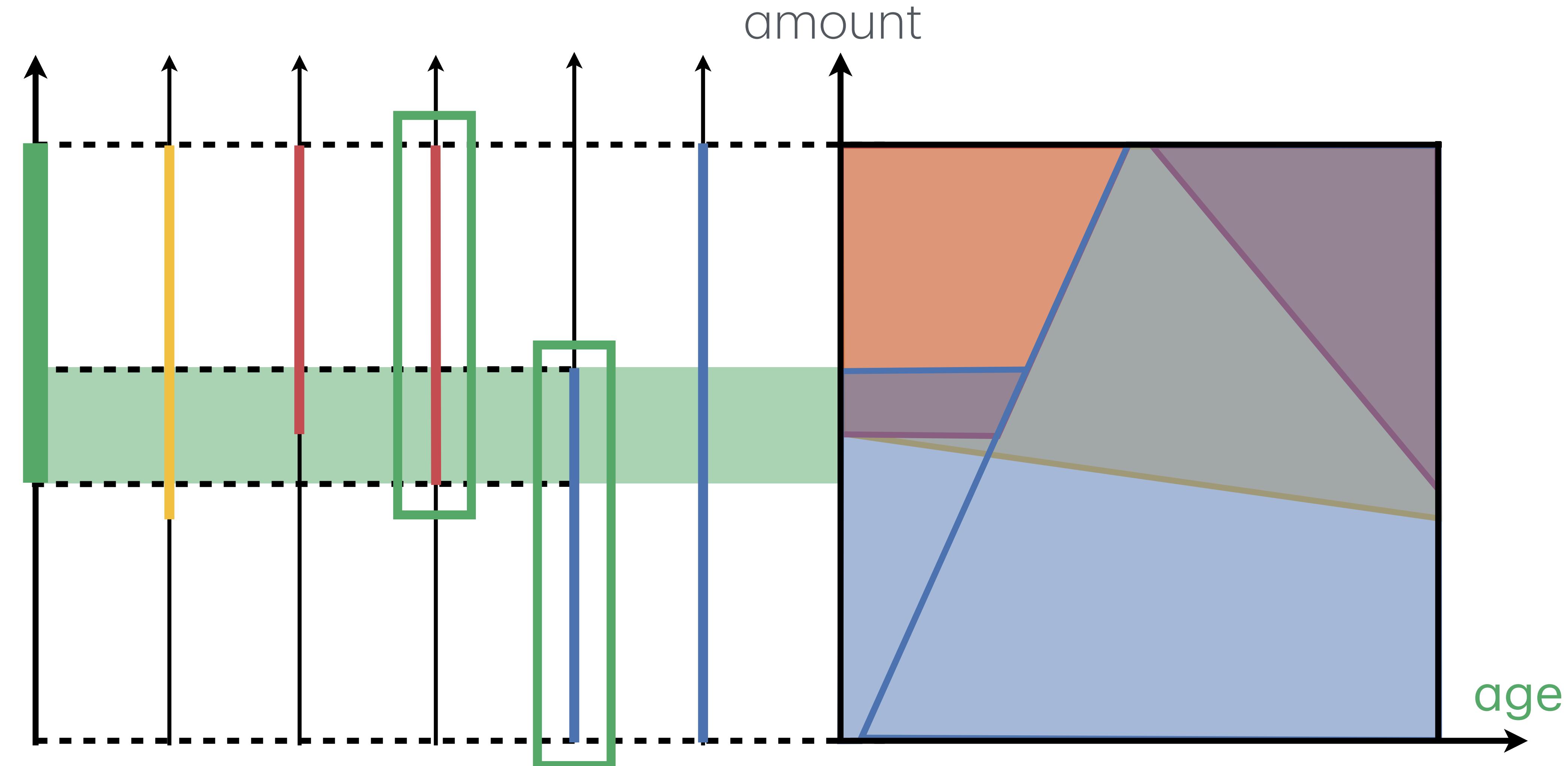
### (iii) Fair Input Space: QAUNUSED



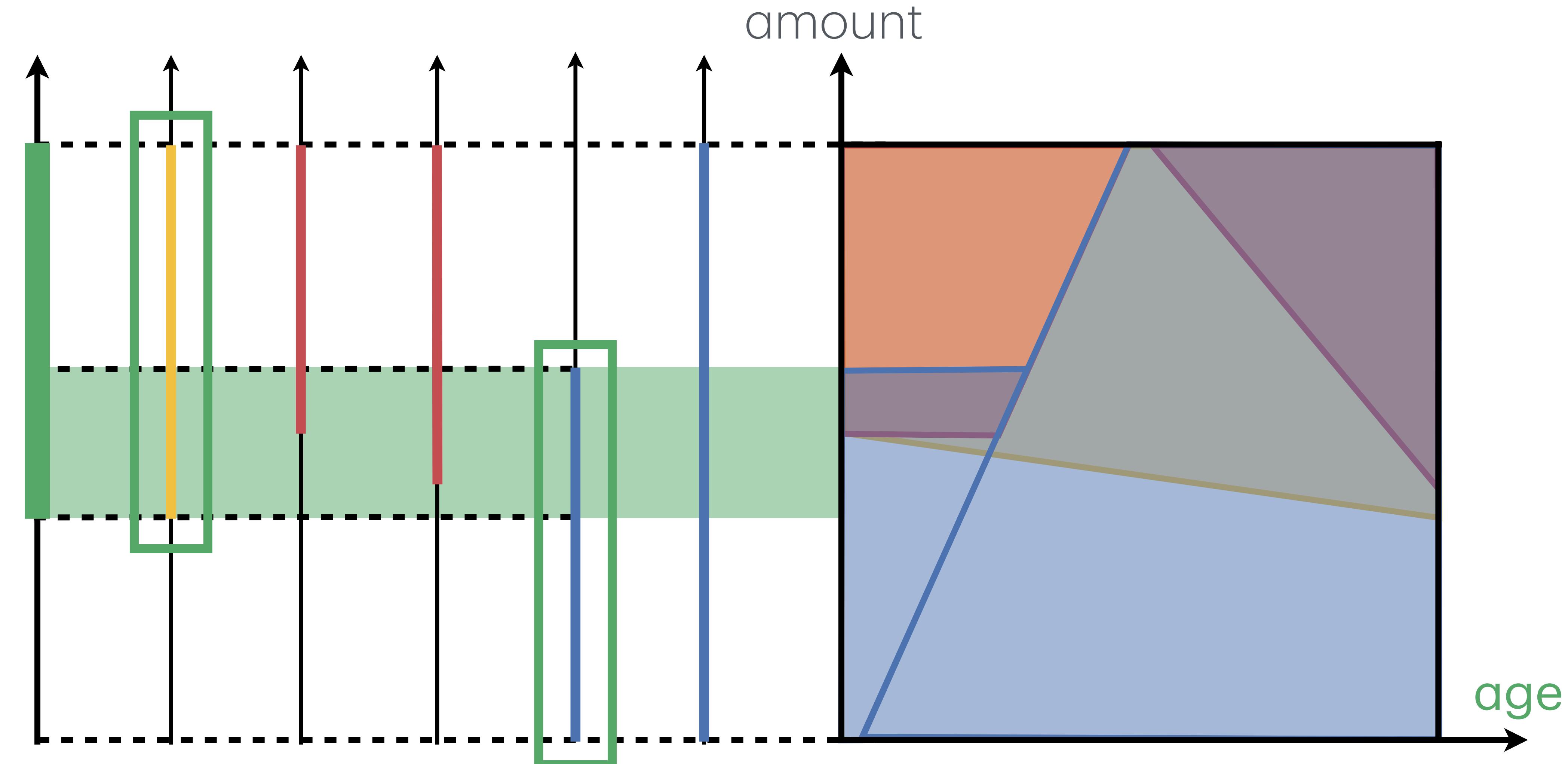
### (iii) Fair Input Space: QAUNUSED



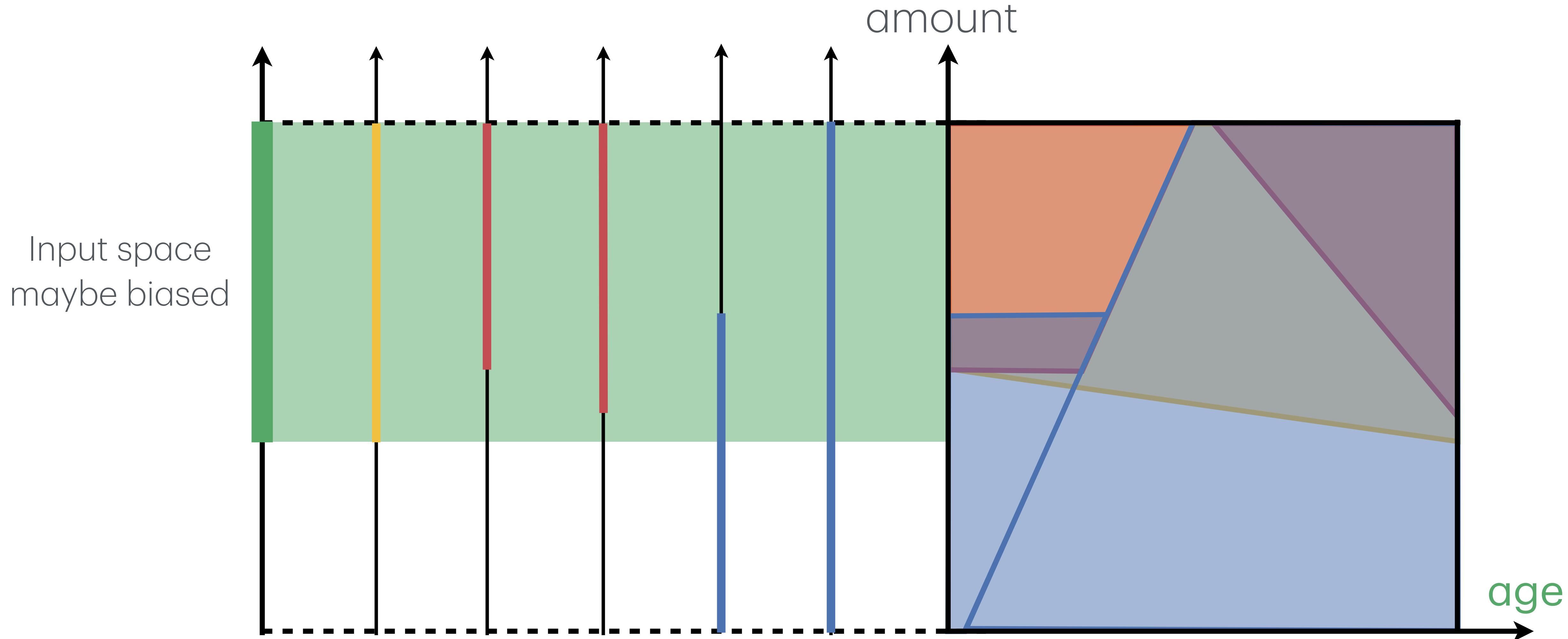
### (iii) Fair Input Space: QAUNUSED



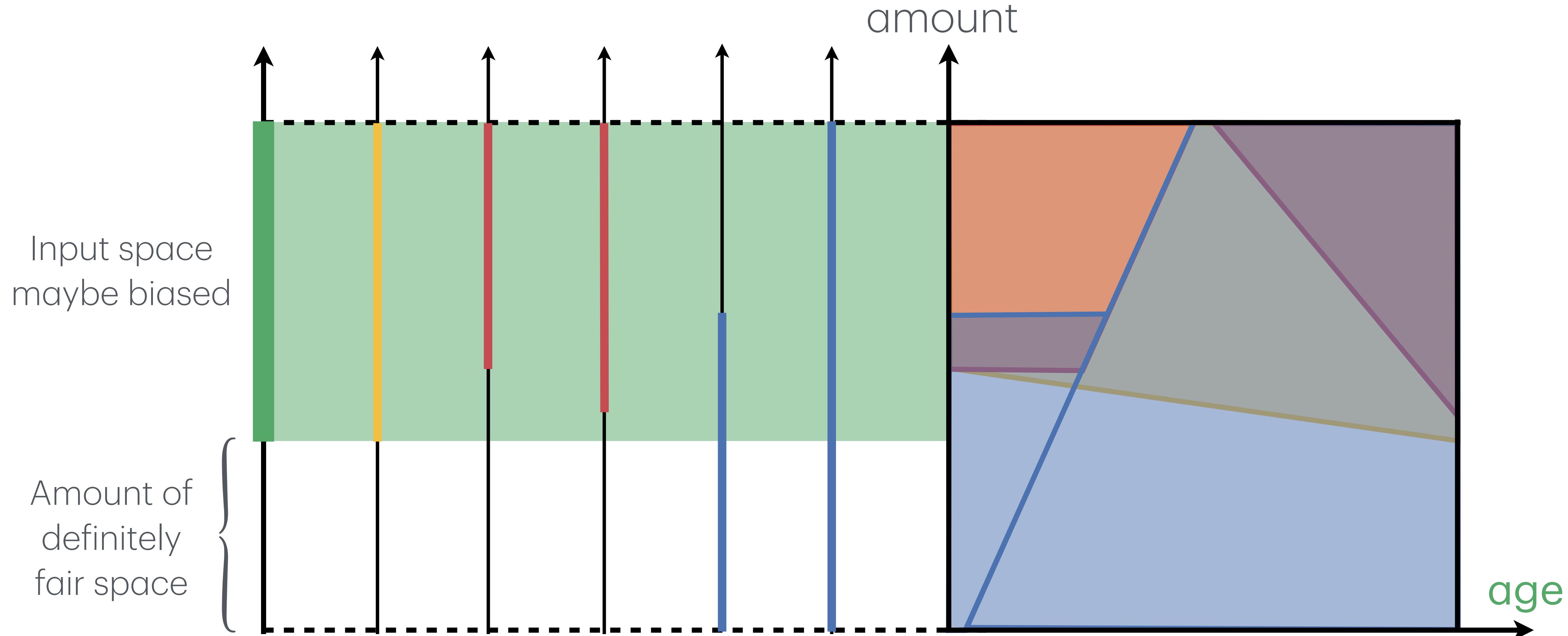
### (iii) Fair Input Space: QAUNUSED



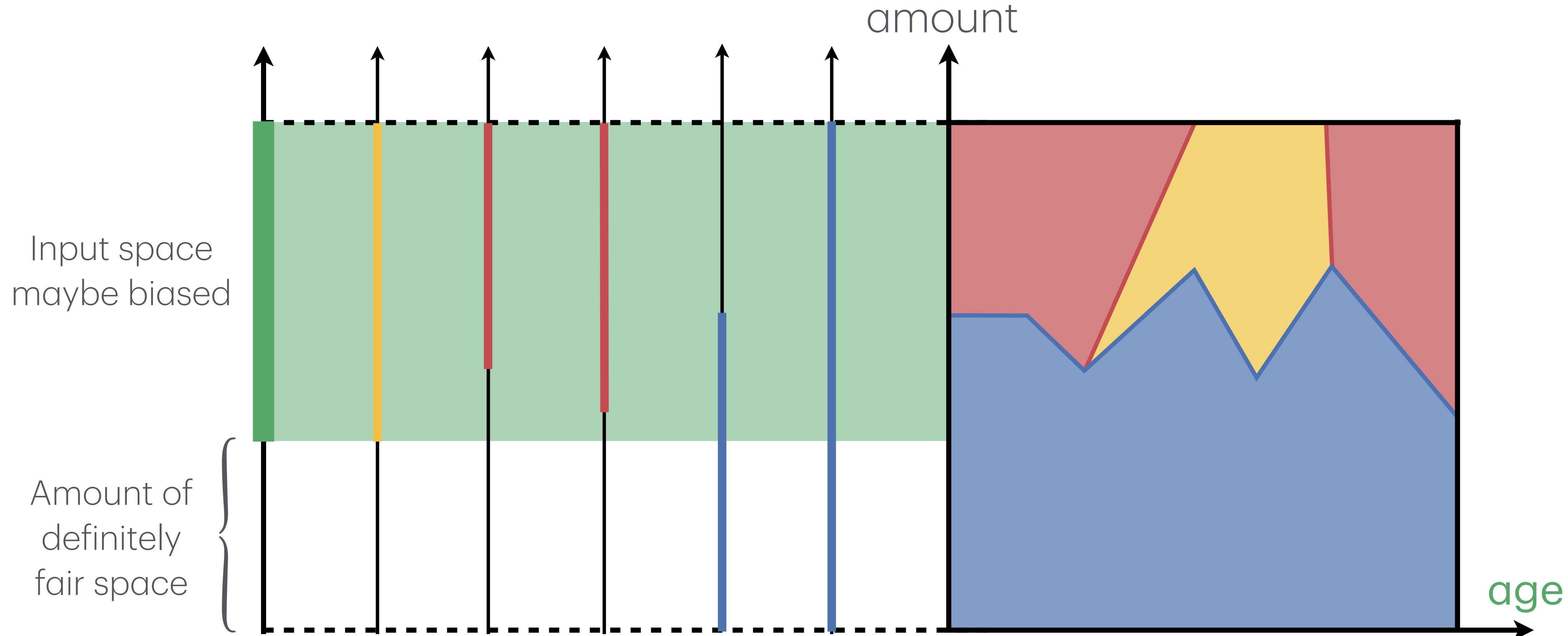
### (iii) Fair Input Space: QAUNUSED



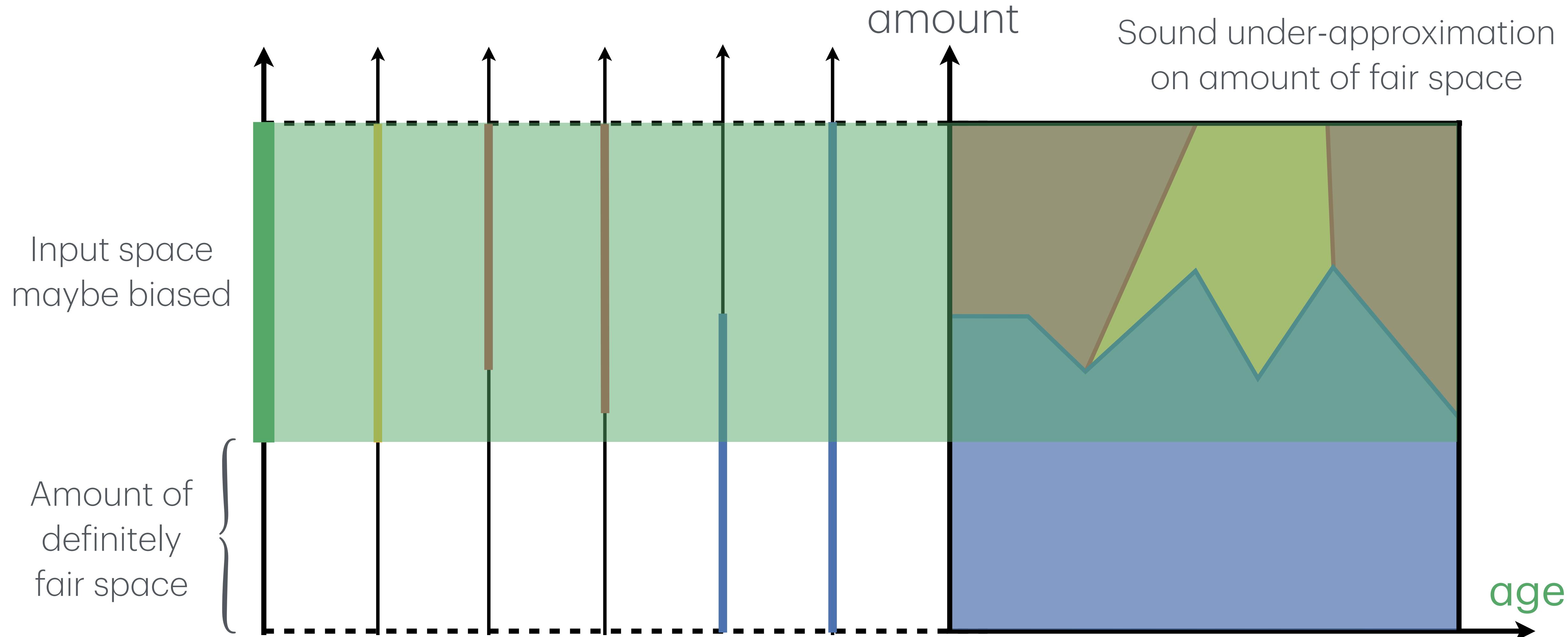
### (iii) Fair Input Space: QAUNUSED



### (iii) Fair Input Space: QAUNUSED



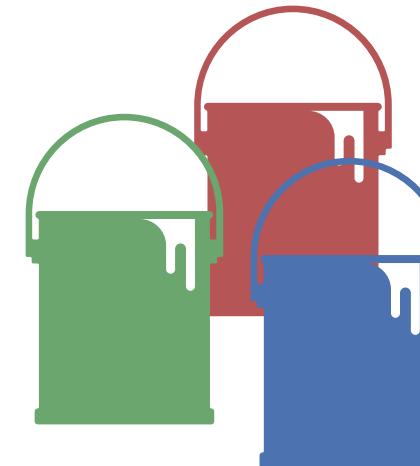
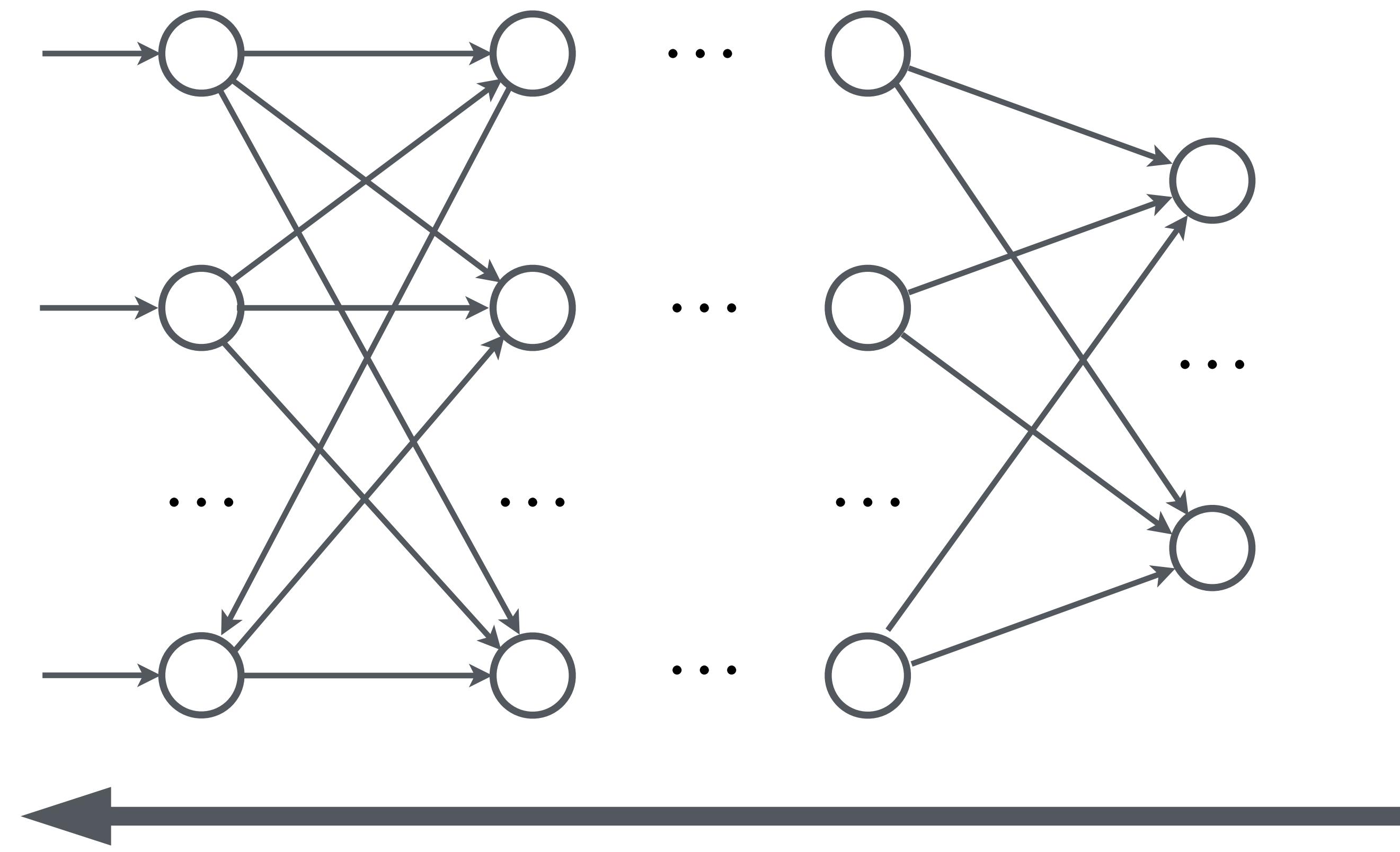
### (iii) Fair Input Space: QAUNUSED



## (ii) Backward Abstract Analysis

Highly precise

Quite slow

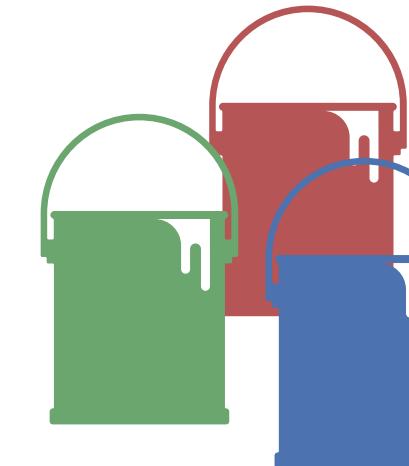
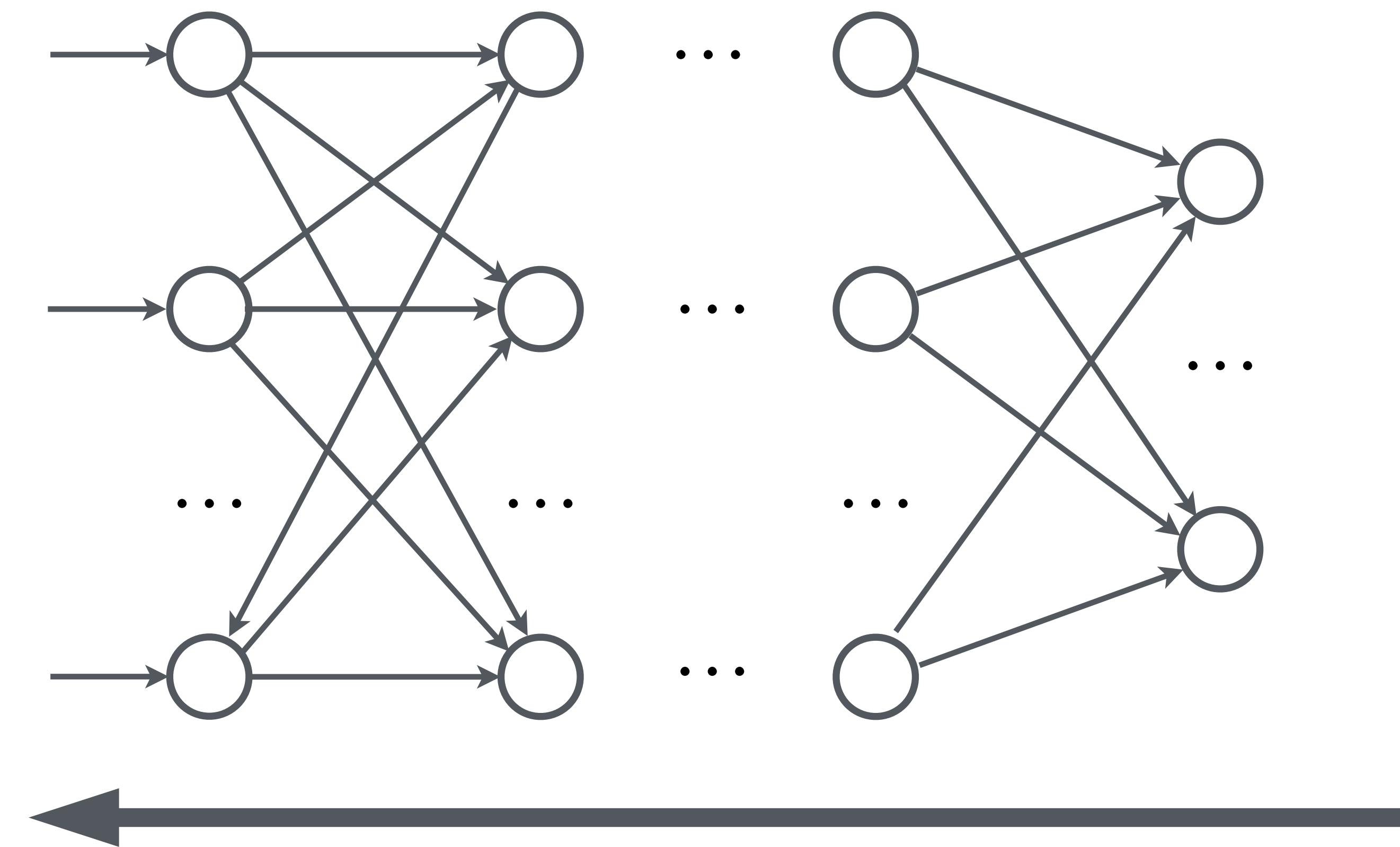


# (ii) Backward Abstract Analysis

Forward pre-analysis to improve scalability

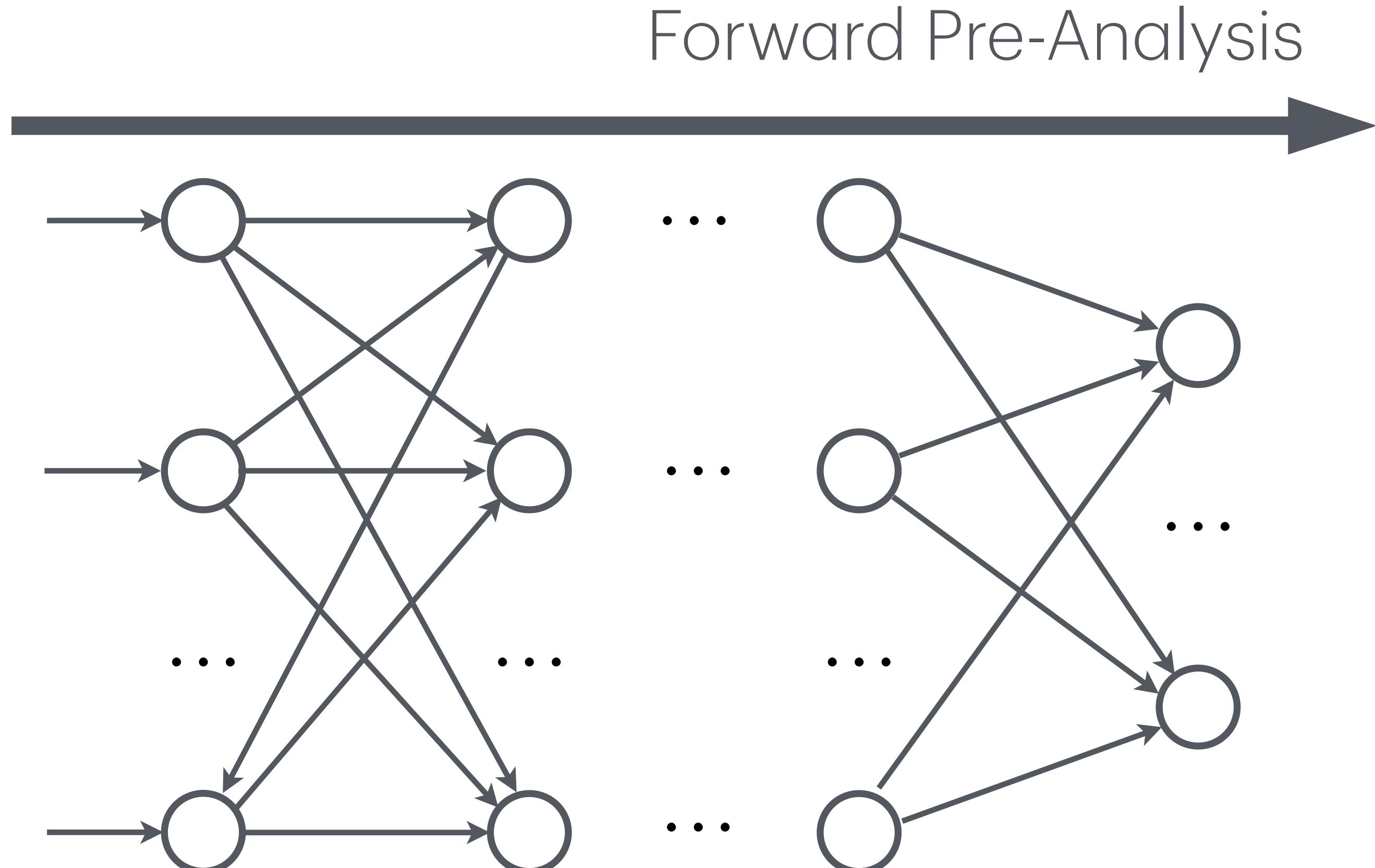
Highly precise

Quite slow



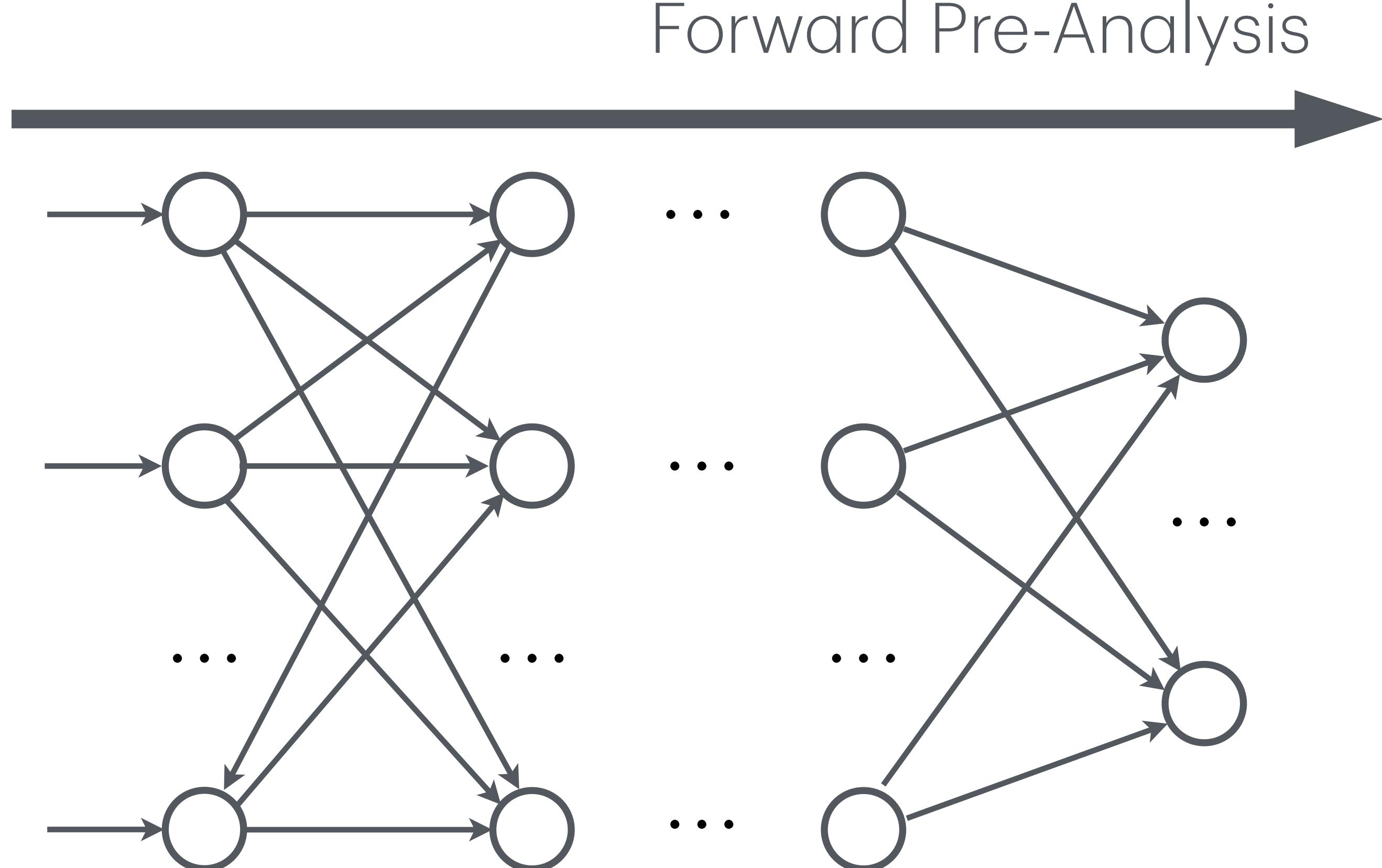
# Forward Pre-Analysis

Via Abstract Interpretation



# Forward Pre-Analysis

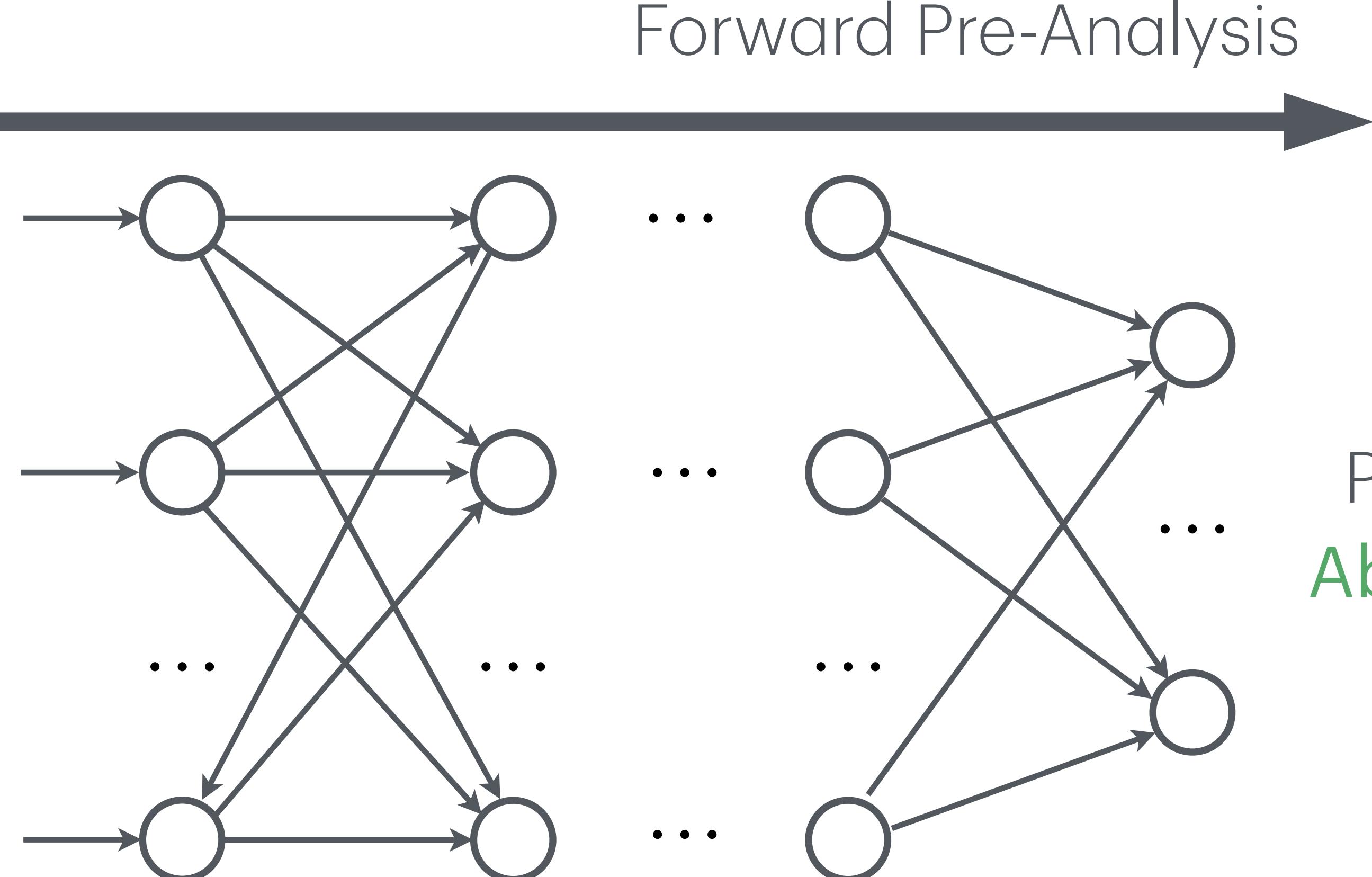
Via Abstract Interpretation



Goal: Reduce the number of paths explored during the backward analysis

# Forward Pre-Analysis

Via Abstract Interpretation



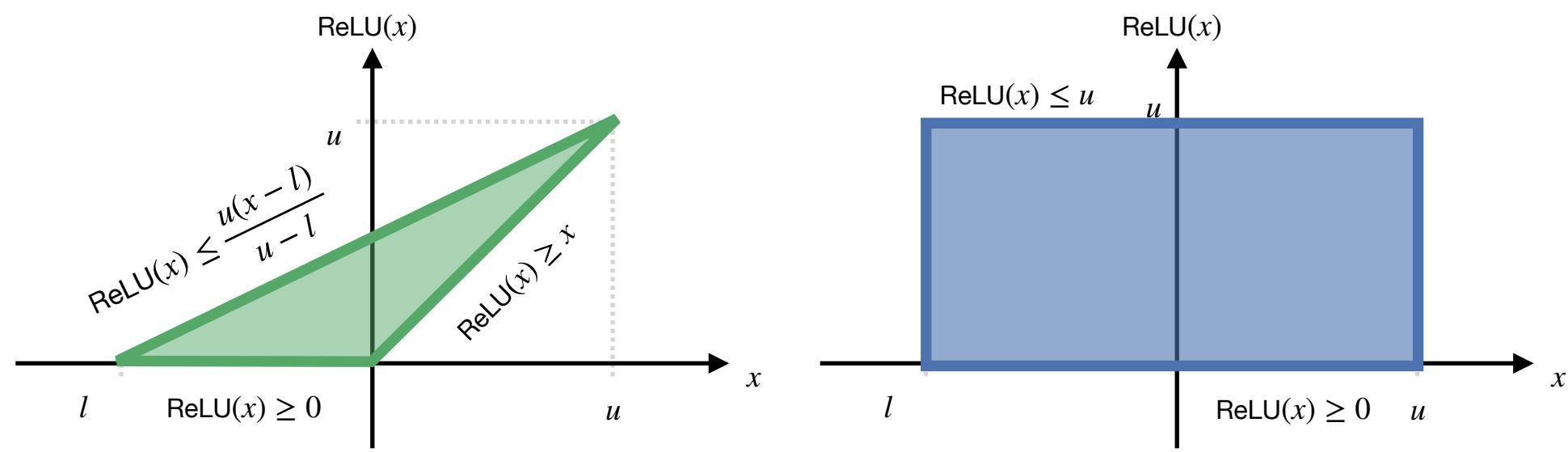
Goal: Reduce the number of paths explored during the backward analysis

# Abstract Domains

## Symbolic

Li et al. @ SAS 2019

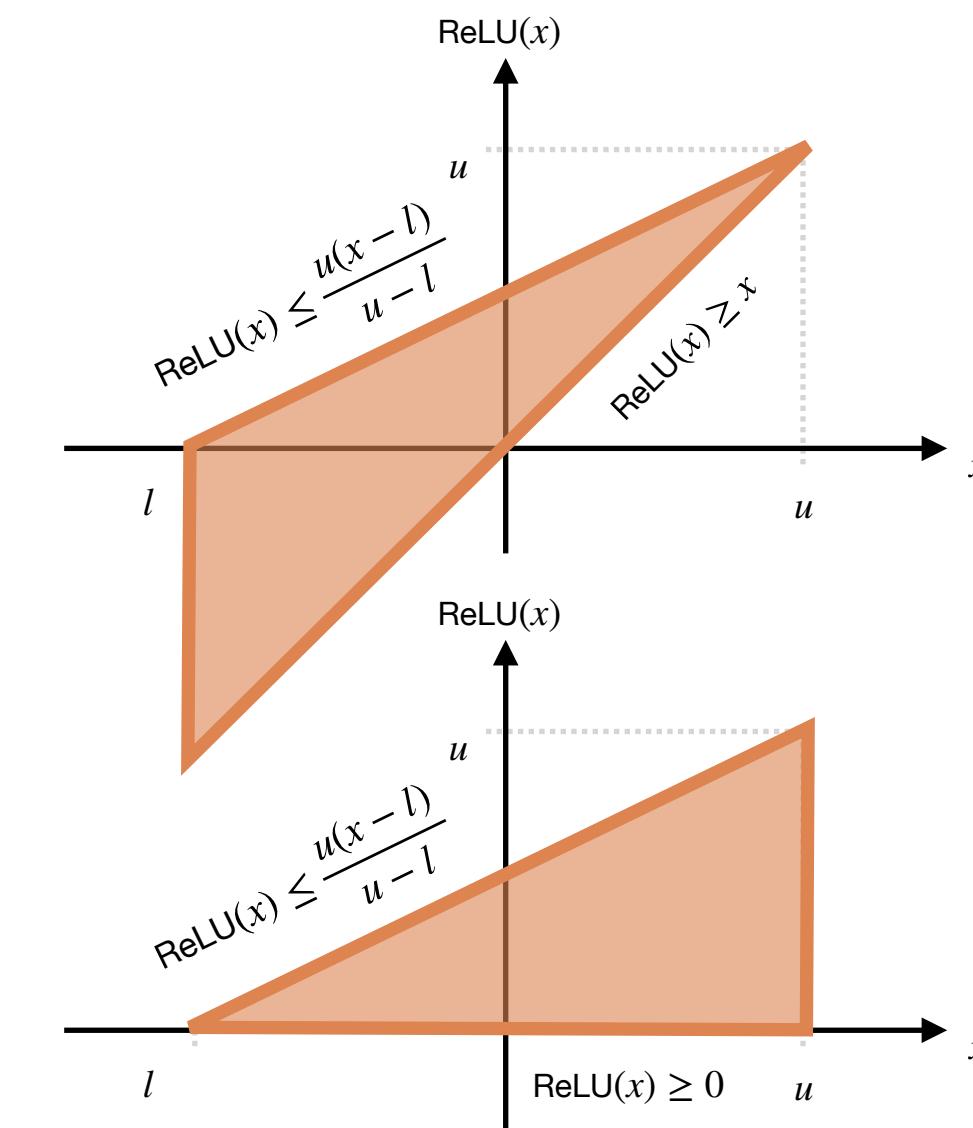
$$[l, u]$$
$$\sum_k m_k \cdot x_k + q$$



## DeepPoly

Singh et al. @ POPL 2019

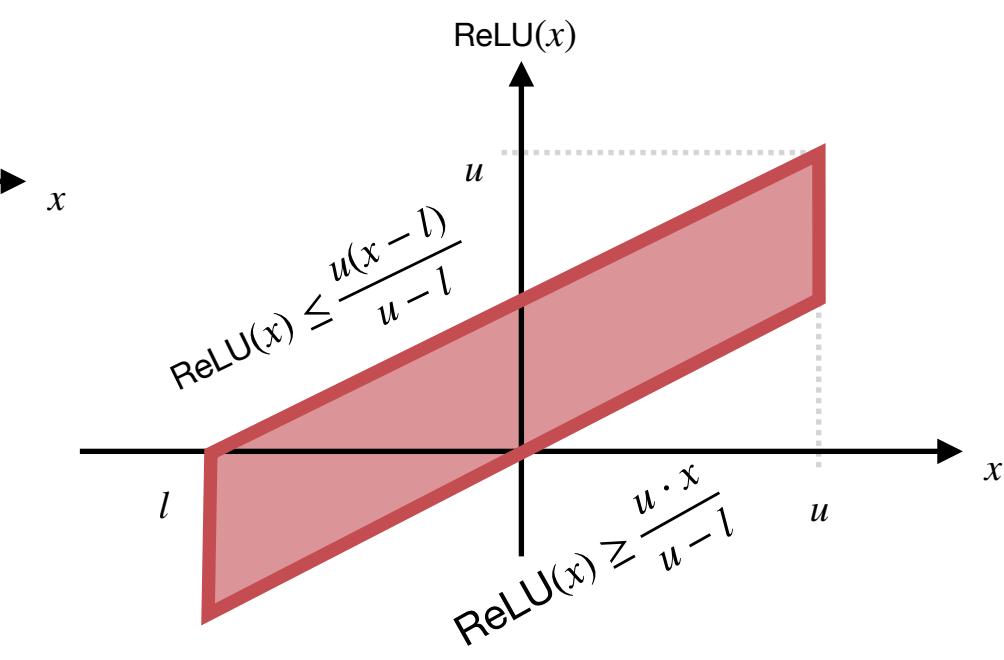
$$[l, u]$$
$$[\text{eq}_{\text{low}}, \text{eq}_{\text{up}}]$$

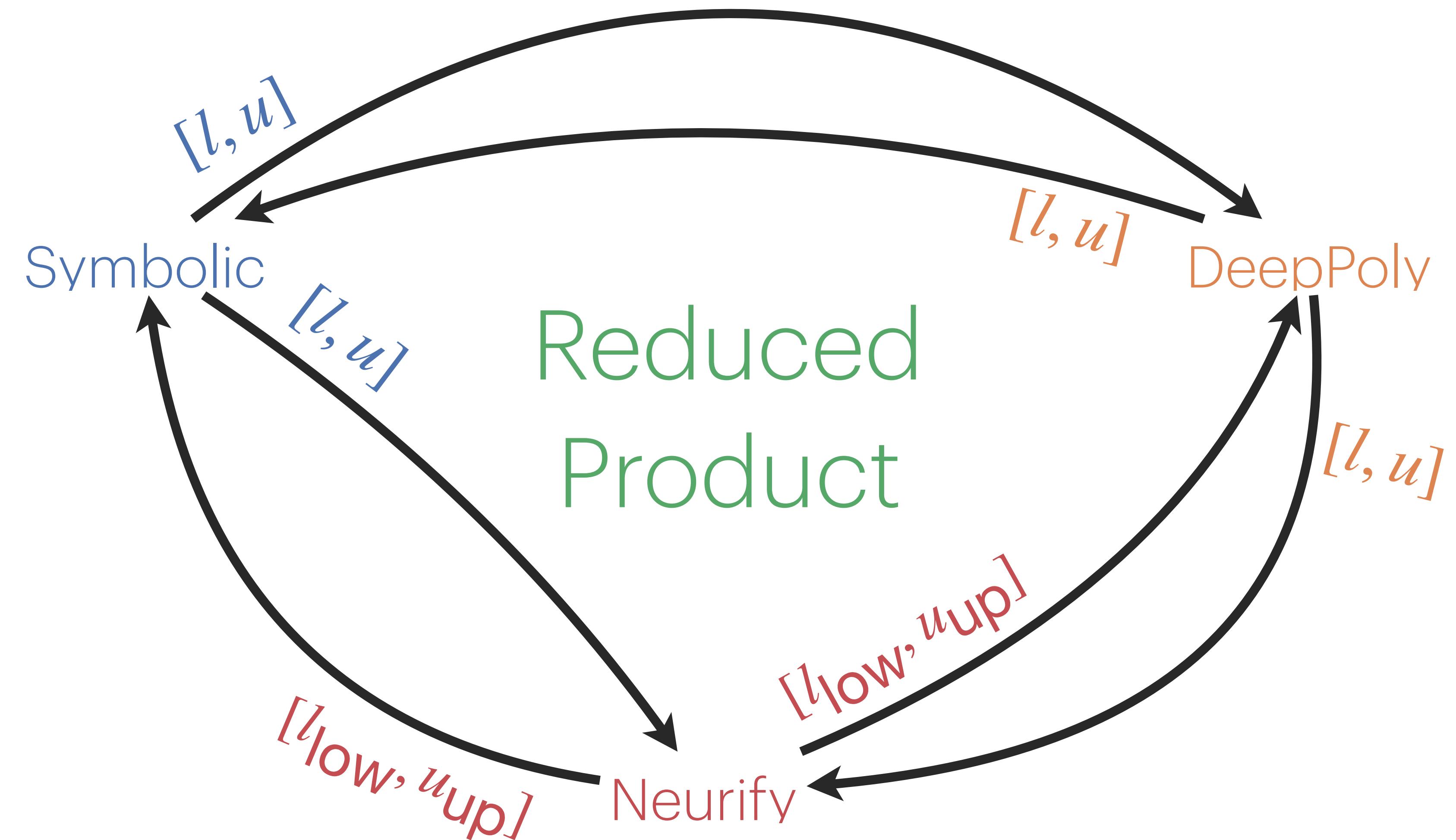


## Neurify

Wang et al. @ NeurIPS 2018

$$[l_{\text{low}}, l_{\text{up}}, u_{\text{low}}, u_{\text{up}}]$$
$$[\text{eq}_{\text{low}}, \text{eq}_{\text{up}}]$$





# Precision-vs-Scalability

L	U	Symbolic	DeepPoly	Neurify	Product	
0.5	3	48.78%	49.01%	46.49%	59.20%	+10.3%
	5	56.11%	56.15%	53.06%	68.23%	+11.9%
0.25	3	83.63%	81.82%	81.40%	87.04%	+3.4%
	5	91.67%	91.58%	92.33%	95.48%	+3.2%



# Reduced Products of Abstract Domains for Fairness Certification of Neural Networks

## Denis Mazzucato and Caterina Urban

SAS 2021

### Reduced Products of Abstract Domains for Fairness Certification of Neural Networks

Denis Mazzucato<sup>[0000-0002-3613-2035]</sup> and Caterina Urban<sup>[0000-0002-8127-9642]</sup>

Inria & École Normale Supérieure | Université PSL  
`{denis.mazzucato,caterina.urban}@inria.fr`

**Abstract.** We present LIBRA, an open-source abstract interpretation-based static analyzer for certifying fairness of ReLU neural network classifiers for tabular data. LIBRA combines a sound forward pre-analysis with an exact backward analysis that leverages the polyhedra abstract domain to provide definite fairness guarantees when possible, and to otherwise quantify and describe the biased input space regions. The analysis is configurable in terms of scalability and precision. We equipped LIBRA with new abstract domains to use in the pre-analysis, including a generic reduced product domain construction, as well as search heuristics to find the best analysis configuration. We additionally set up the backward analysis to allow further parallelization. Our experimental evaluation demonstrates the effectiveness of the approach on neural networks trained on a popular dataset in the fairness literature.

**Keywords:** Fairness · Neural Networks · Reduced Abstract Domain Products · Abstract Interpretation · Static Analysis



#### 1 Introduction

Nowadays, machine learning software has an ever increasing societal impact by assisting or even automating decision making in fields such as social welfare, criminal justice, and even health care. At the same time, a number of recent cases have shown that such software may reproduce, or even reinforce, bias directly or indirectly present in the training data [3,16,17,23]. In April 2021, the European Commission proposed a first legal framework on machine learning software – the Artificial Intelligence Act [10] — which imposes strict requirements to minimize the risk of discriminatory outcomes. In this context, methods and tools for certifying fairness or otherwise detecting bias are extremely valuable.

# Extensional Properties

## Neural Networks

# Theoretical Framework

---

## Extensional Properties

---

Neural  
Networks

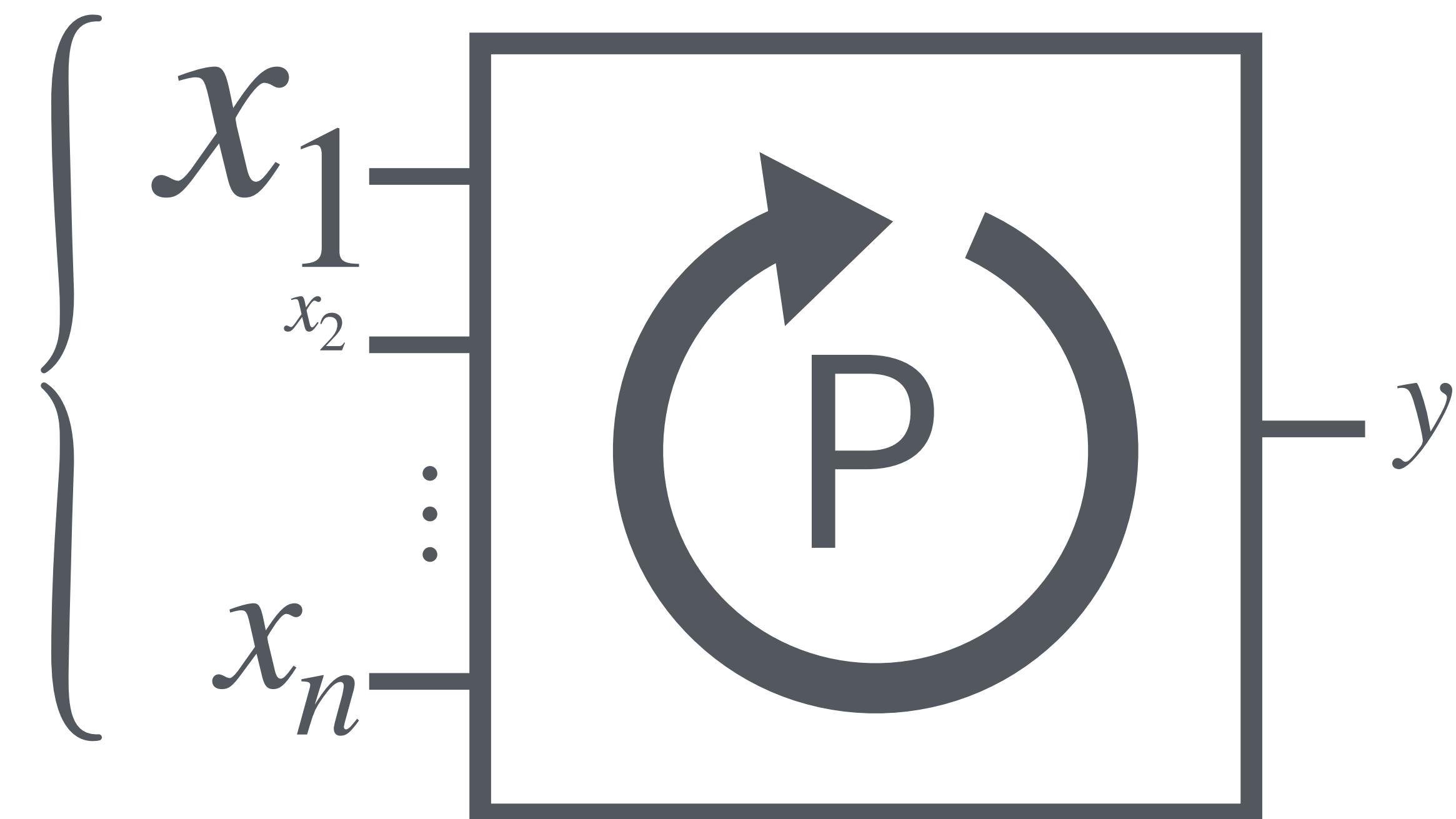
## Intensional Properties

---

Loop  
Iterations

# Why

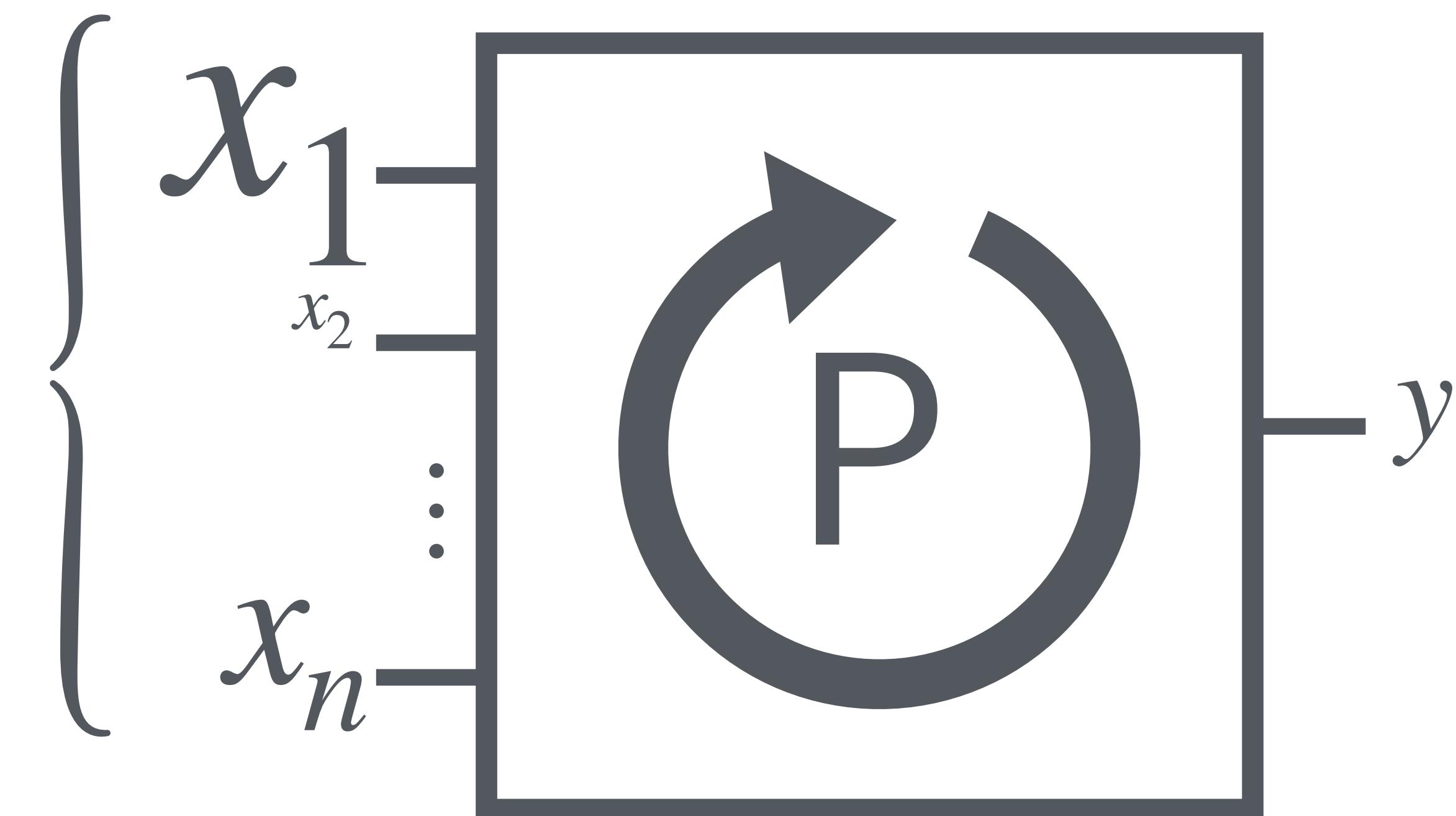
---



# Why

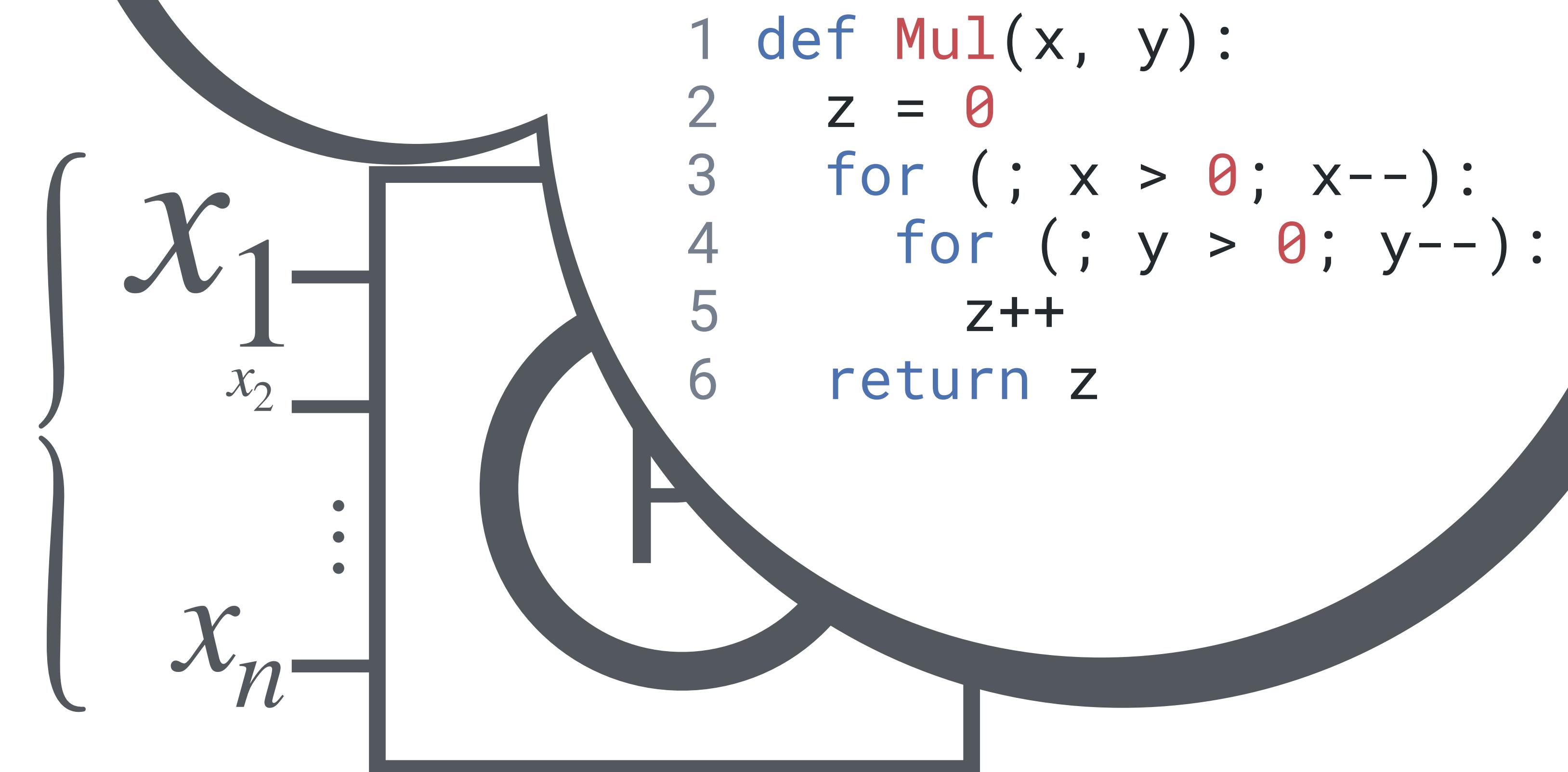
---

- Correct Loop Behavior



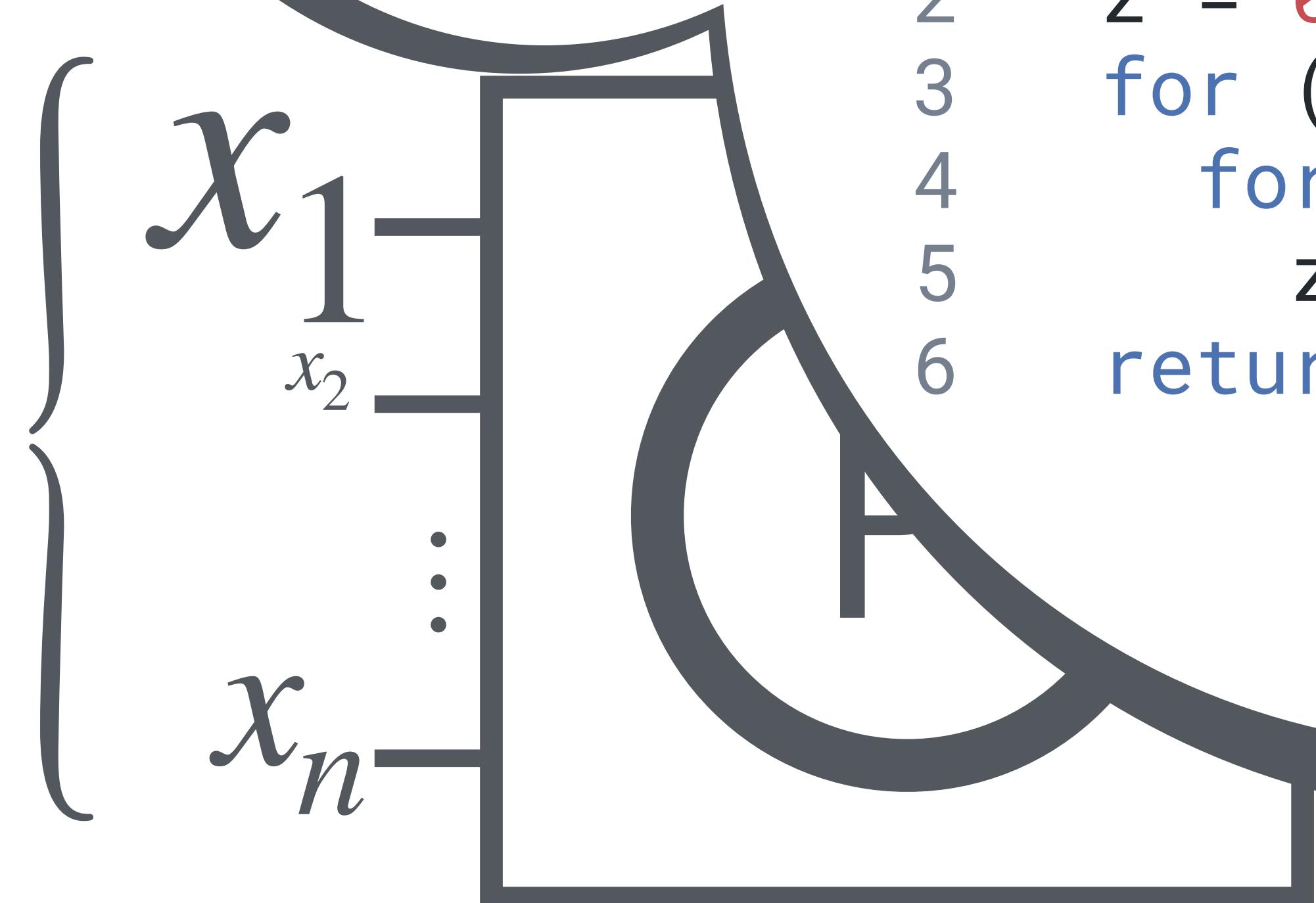
# Why

- Correct Loop Behavior



# Why

- Correct Loop Behavior



$$\text{IMPACT}_x(\text{Mul}) = 2 \cdot \text{IMPACT}_y(\text{Mul})$$

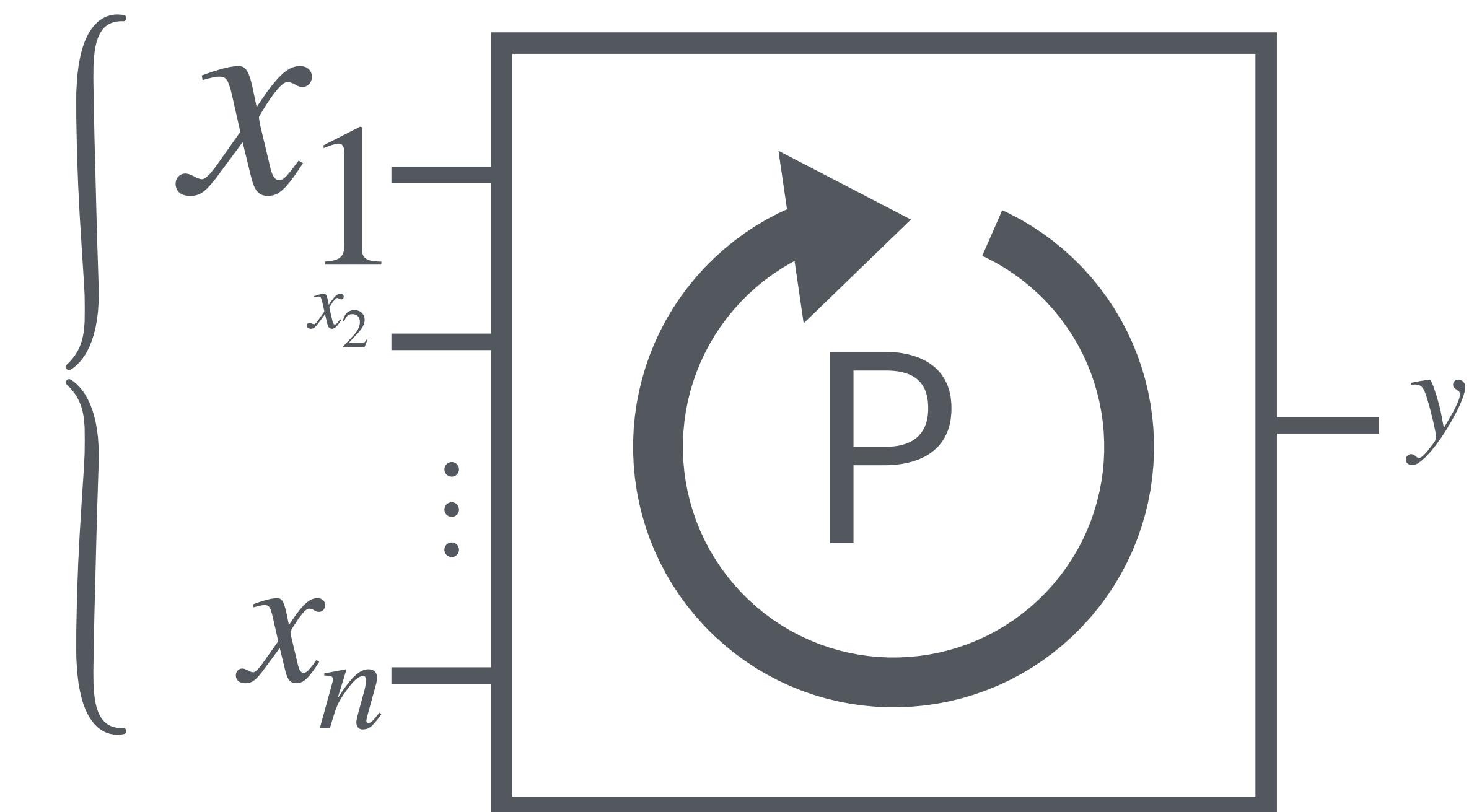


```
1 def Mul(x, y):  
2     z = 0  
3     for (; x > 0; x--):  
4         for (; y > 0; y--):  
5             z++  
6     return z
```

# Why

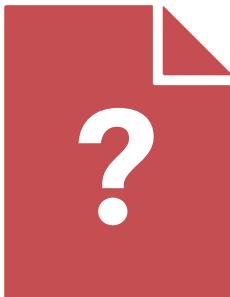
---

- Correct Loop Behavior
- Performance



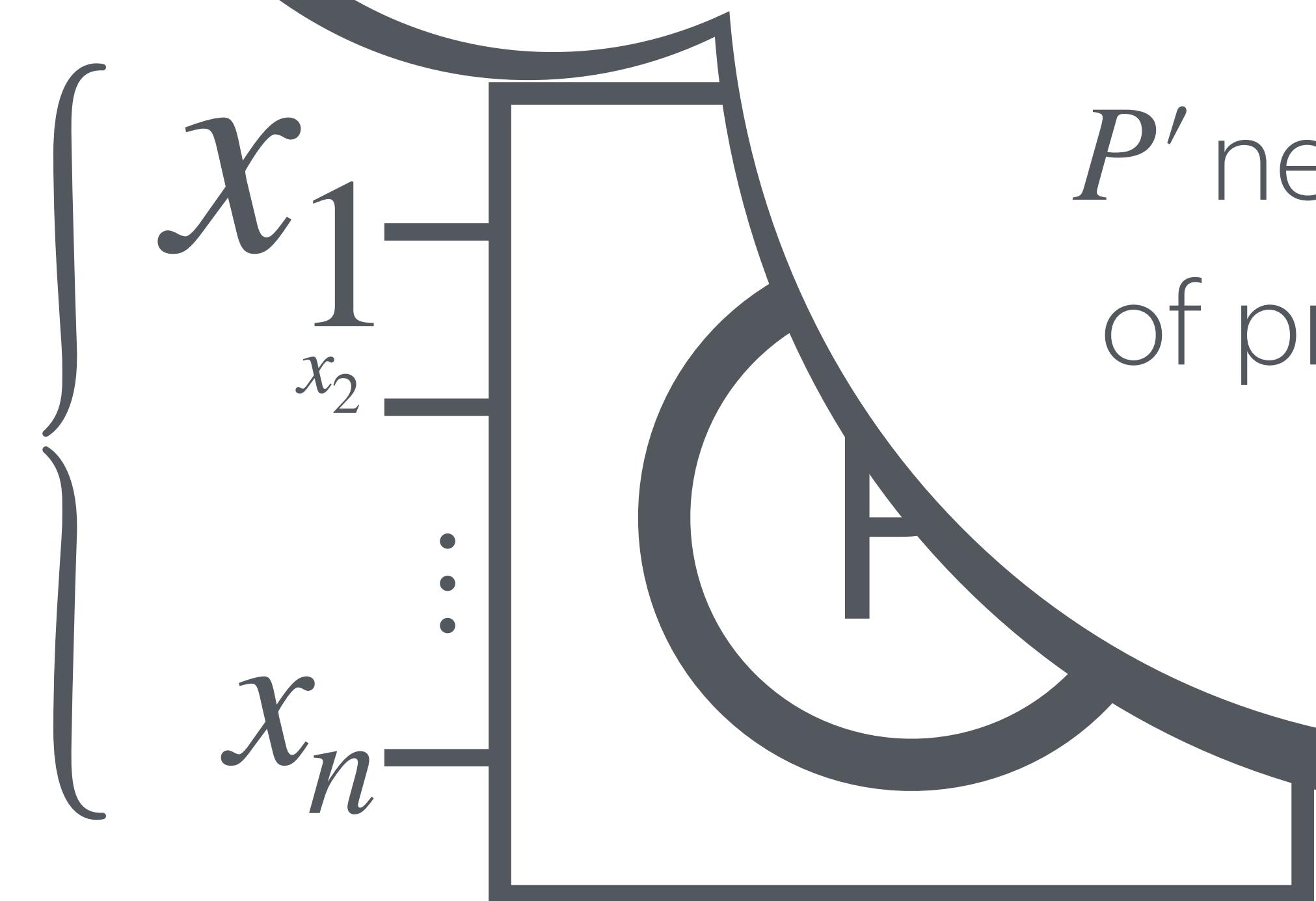
# Why

$$\text{IMPACT}_x(P) \leq \text{IMPACT}_x(P')$$



Regression!

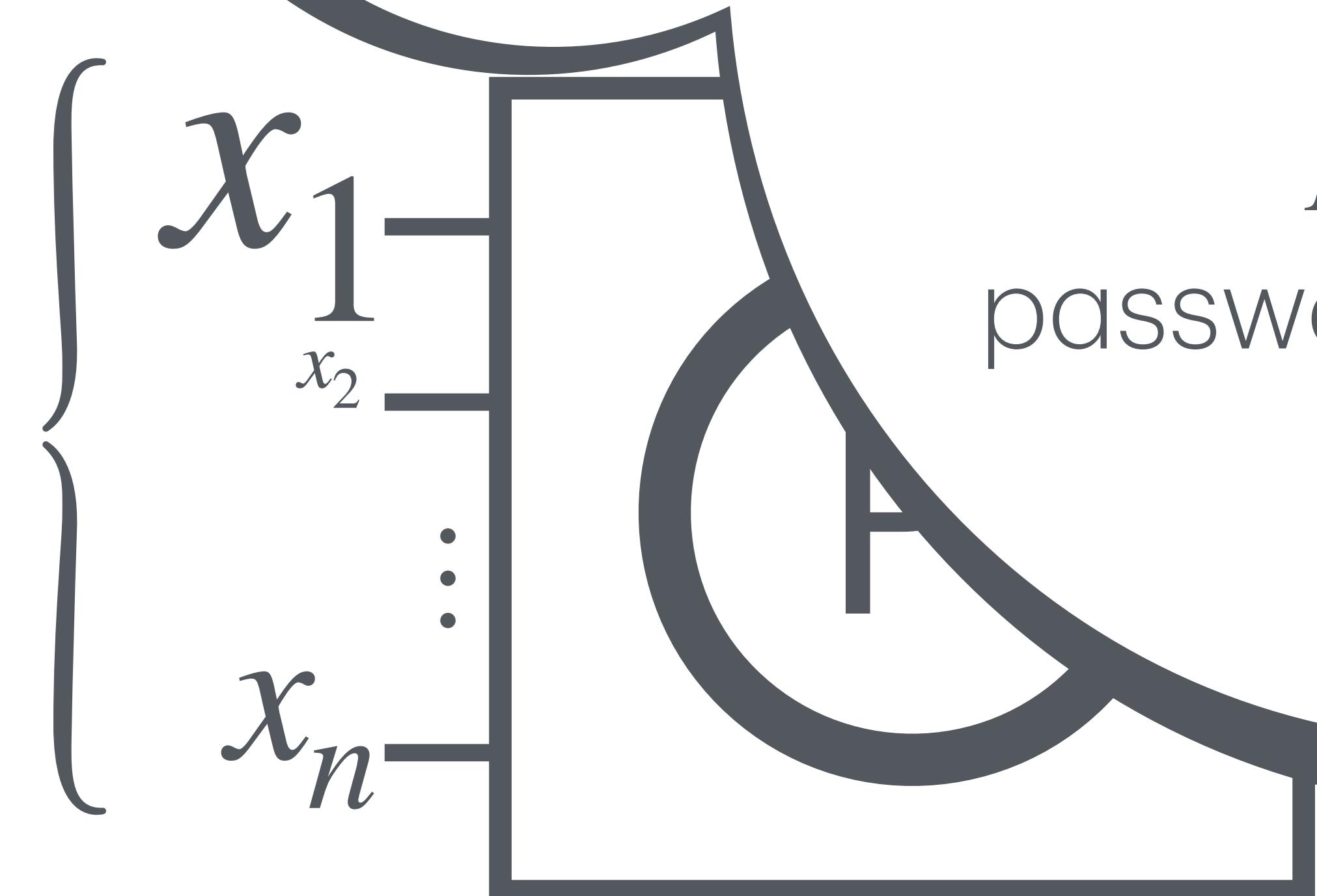
- Correct Loop Behavior
- Performance



$P'$  new version  
of program  $P$

# Why

- Correct Loop Behavior
- Performance
- Security



$\text{IMPACT}_{\text{pwd}}(P) > 0$



Timing side-channels!

$P$  is a  
password checker

# Add Function

---

$$\begin{array}{r} 4 \ 2 = \\ \hline 3 \ 8 + \\ 4 \end{array}$$

# Add Function

---

array

$$\frac{[0, 4, 2] =}{[3, 8] + [4]}$$

# Add Function

---

length      array

$$\begin{array}{r} 3 \underline{[0, 4, 2]} = \\ 2 \quad [3, 8] + \\ 1 \quad [4] \end{array}$$

# Add Function

length array

$$\begin{array}{r} 3 \\ \underline{-} \\ 2 \\ \underline{-} \\ 1 \end{array} [0, 4, 2] = [3, 8] + [4]$$

from: s2n-bignum AWS

```
1 def Add(p, z, m, x, n, y): 26
2     r = min(p, m)                27
3     s = min(p, n)                28
4     if (r < s):                 29
5         t = p - s                30
6         q = s - r                31
7         i = 0                     32
8         a = 0                     33
9         for (; r > 0; r--):      34
10            s = x[i]              35
11            w = y[i]              36
12            z[i] = s + w + a    37
13            i = i + 1             38
14            a = (w < a) ||       39
15            (s + w < s) ||       40
16            (s + w + a < s)      41
17        do:                      42
18            r = y[i]              43
19            b = (r < a) ||       44
20            (r + a < r)          45
21            z[i] = r + a          46
22            i = i + 1             47
23            q--                   48
24            a = b                  49
25        while (q > 0):          50
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
else:
    t = p - r
    q = r - s
    i = 0
    b = 0
    for (; s > 0; s--):
        r = x[i]
        w = y[i]
        z[i] = r + w + b
        i = i + 1
        b = (w < b) ||
             (r + w < r) ||
             (r + w + b < r)
    for (; q > 0; q--):
        r = x[i]
        z[i] = r + b
        i = i + 1
        b = (r < b) ||
             (r + b < r)
    if (t > 0):
        z[i] = b
    while (t > 0):
        i = i + 1
        t--
        if (t > 0):
            z[i] = 0
```

# The RANGE Quantifier

---

# The RANGE Quantifier

---

$W = p$

p	z	m	x	n	y	
Add(	3,	z,	2,	[4,2],	2,	[6,8] ) = z ↦ [1,1,0]

# The RANGE Quantifier

---

	p	z	m	x	n	y	W = p
Add(	3,	z,	2,	[4,2],	2,	[6,8]	) = z ↦ [1,1,0]
Add(	2,	z,	2,	[4,2],	2,	[6,8]	) = z ↦ [1,0]

# The RANGE Quantifier

	p	z	m	x	n	y	W = p
Add(	3,	z,	2,	[4,2],	2,	[6,8]	) = z ↦ [1,1,0]
Add(	2,	z,	2,	[4,2],	2,	[6,8]	) = z ↦ [1,0]
Add(	4,	z,	2,	[4,2],	2,	[6,8]	) = z ↦ [0,1,1,0]

# The RANGE Quantifier

	p	z	m	x	n	y	W = p
Add(	3,	z,	2,	[4,2],	2,	[6,8]	) = z ↪ [1,1,0]
Add(	2,	z,	2,	[4,2],	2,	[6,8]	) = z ↪ [1,0]
Add(	4,	z,	2,	[4,2],	2,	[6,8]	) = z ↪ [0,1,1,0]

Missing information about the number of iterations

# The RANGE Quantifier

---

p z m x n y

W = p

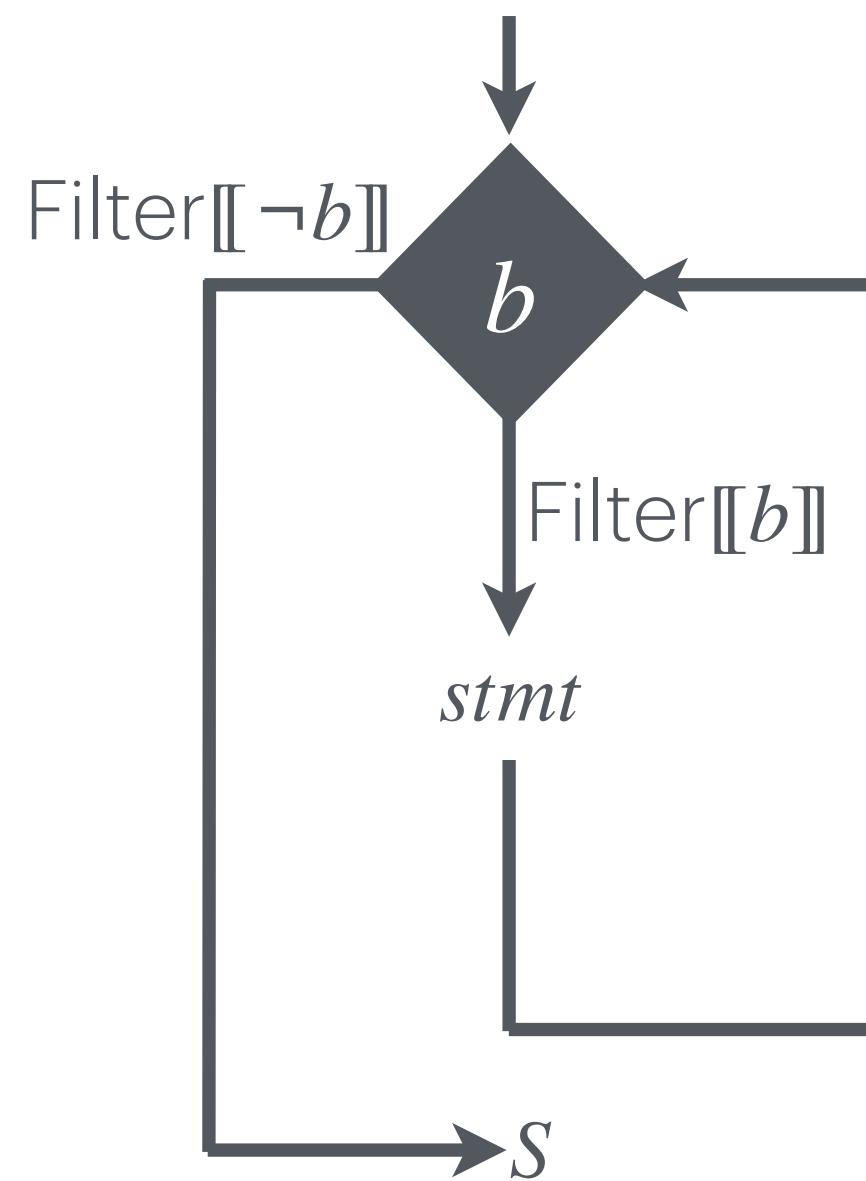
Add( 3, z, 2, [4,2], 2, [6,8] ) = z ↦ [1,1,0]

Add( 2, z, 2, [4,2], 2, [6,8] ) = z ↦ [1,0]

Add( 4, z, 2, [4,2], 2, [6,8] ) = z ↦ [0,1,1,0]

Modify the concrete semantics  
to collect the loop counter

# The RANGE Quantifier



p z m x n y

$$\text{Add}(3, z, 2, [4, 2], 2, [6, 8]) = z \mapsto [1, 1, 0]$$

$$\text{Add}(2, z, 2, [4, 2], 2, [6, 8]) = z \mapsto [1, 0]$$

$$\text{Add}(4, z, 2, [4, 2], 2, [6, 8]) = z \mapsto [0, 1, 1, 0]$$

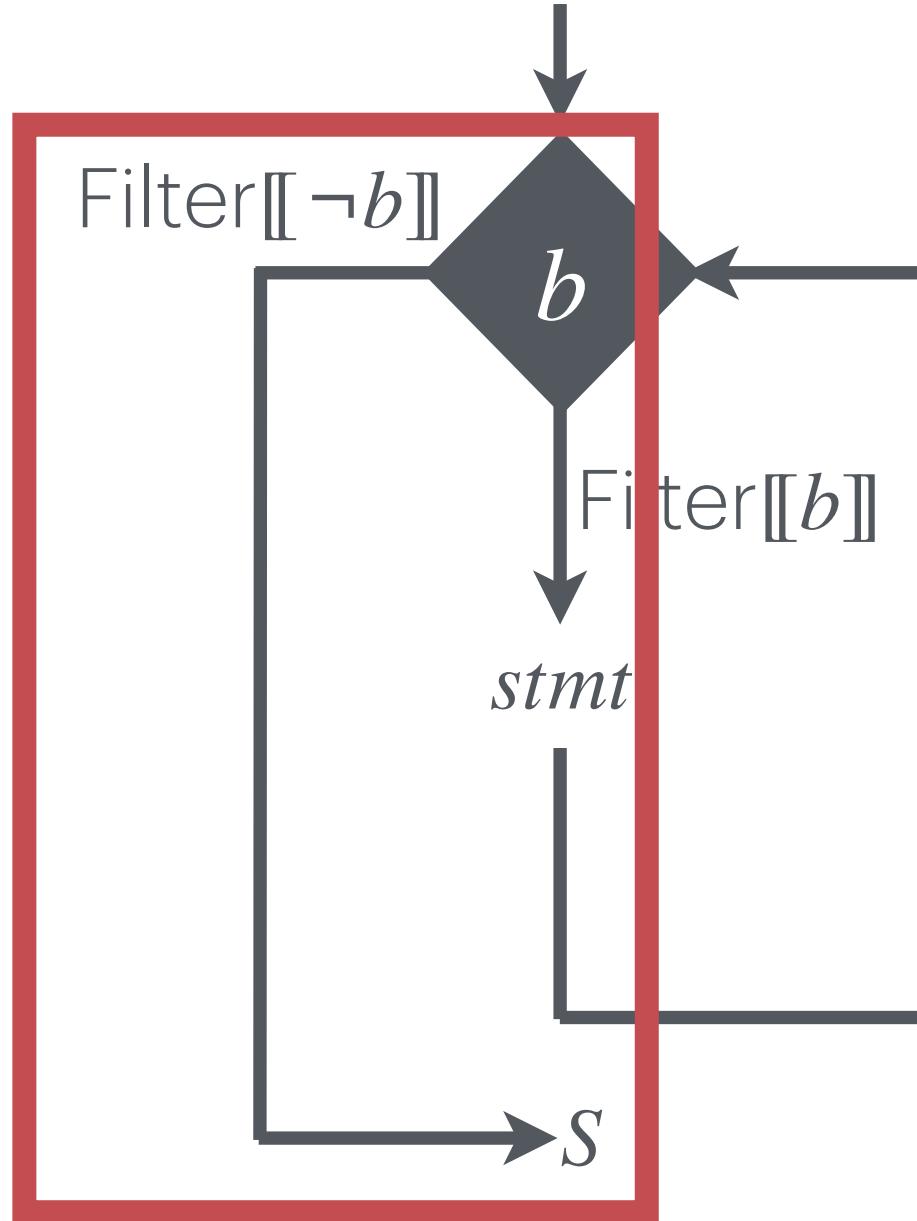
$W = p$

Modify the concrete semantics  
to **collect** the **loop counter**

$$\Lambda[\text{while } b \text{ do } stmt]S \triangleq \lim_n F^n$$

$$F(x) \triangleq \text{Filter}[\neg b]S \cup \text{Filter}[b](\Lambda[stmt](x))$$

# The RANGE Quantifier



p z m x n y

$$\text{Add}( 3, z, 2, [4, 2], 2, [6, 8] ) = z \mapsto [1, 1, 0]$$

$$\text{Add}( 2, z, 2, [4, 2], 2, [6, 8] ) = z \mapsto [1, 0]$$

$$\text{Add}( 4, z, 2, [4, 2], 2, [6, 8] ) = z \mapsto [0, 1, 1, 0]$$

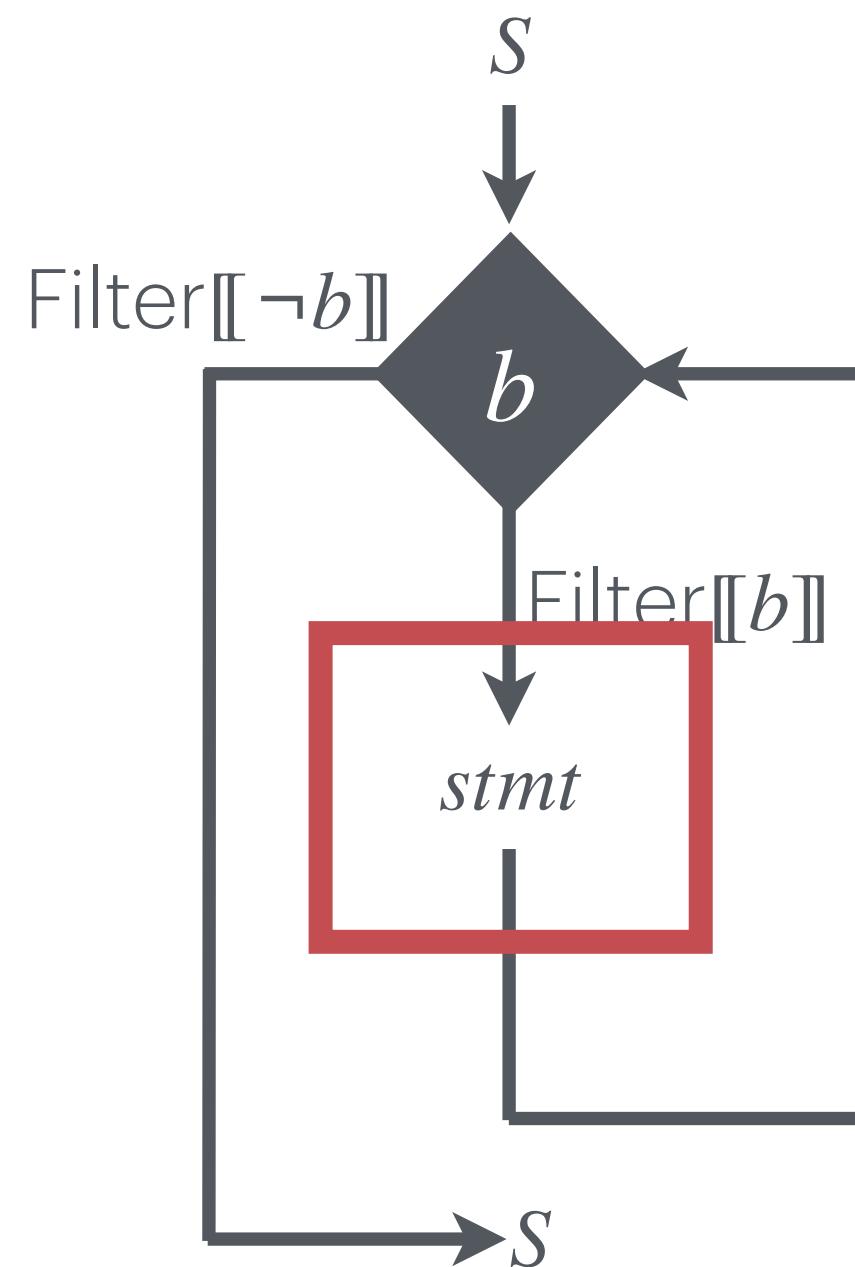
$W = p$

Modify the concrete semantics  
to collect the loop counter

$$\Lambda[\text{while } b \text{ do } stmt]S \triangleq \lim_n F^n$$

$$F(x) \triangleq \text{Filter}[\neg b]S \cup \text{Filter}[b](\Lambda[stmt](x))$$

# The RANGE Quantifier



p z m x n y

$$\text{Add}(3, z, 2, [4, 2], 2, [6, 8]) = z \mapsto [1, 1, 0]$$

$$\text{Add}(2, z, 2, [4, 2], 2, [6, 8]) = z \mapsto [1, 0]$$

$$\text{Add}(4, z, 2, [4, 2], 2, [6, 8]) = z \mapsto [0, 1, 1, 0]$$

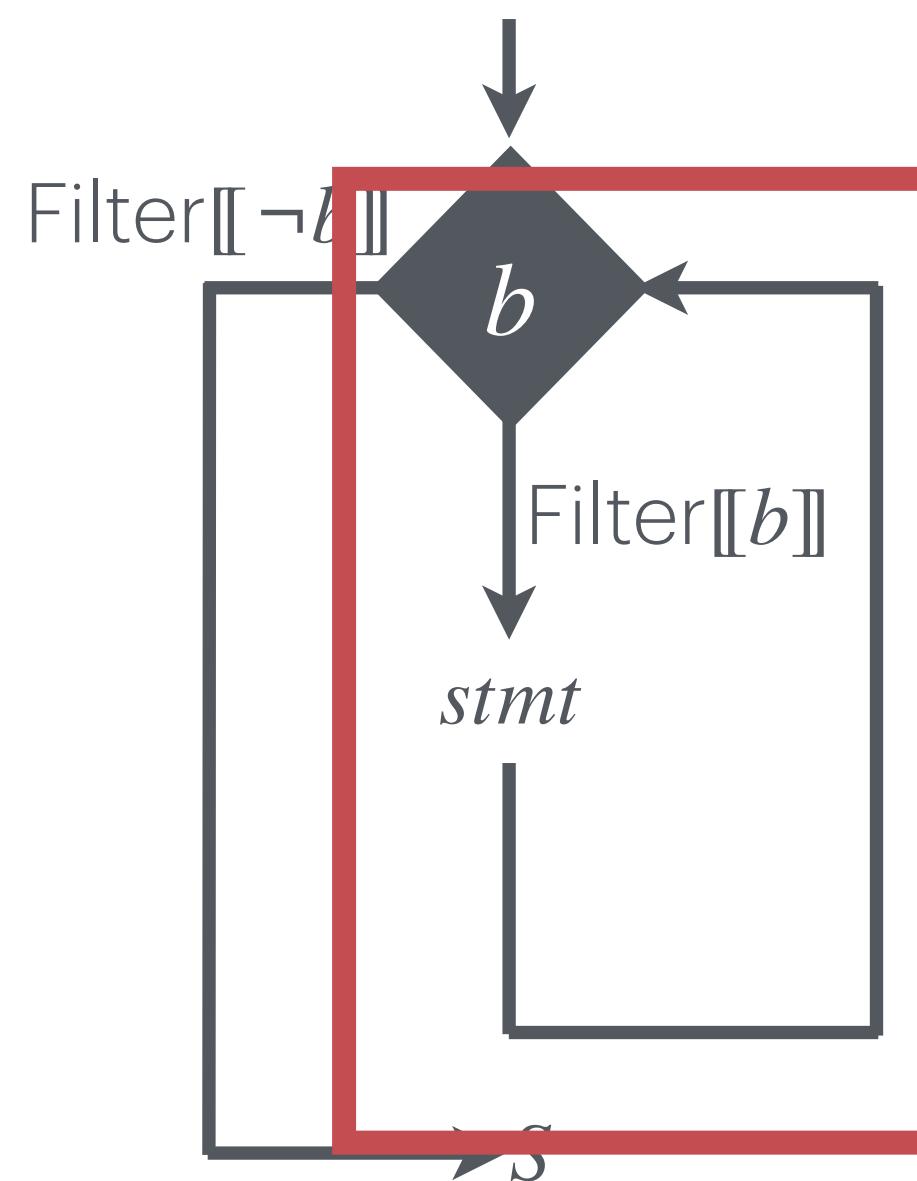
$W = p$

Modify the concrete semantics  
to collect the loop counter

$$\Lambda \llbracket \text{while } b \text{ do } stmt \rrbracket S \triangleq \lim_n F^n$$

$$F(x) \triangleq \text{Filter}\llbracket \neg b \rrbracket S \cup \text{Filter}\llbracket b \rrbracket (\Lambda \llbracket stmt \rrbracket(x))$$

# The RANGE Quantifier



p z m x n y

Add( 3, z, 2, [4,2], 2, [6,8] ) = z ↦ [1,1,0]

Add( 2, z, 2, [4,2], 2, [6,8] ) = z ↦ [1,0]

Add( 4, z, 2, [4,2], 2, [6,8] ) = z ↦ [0,1,1,0]

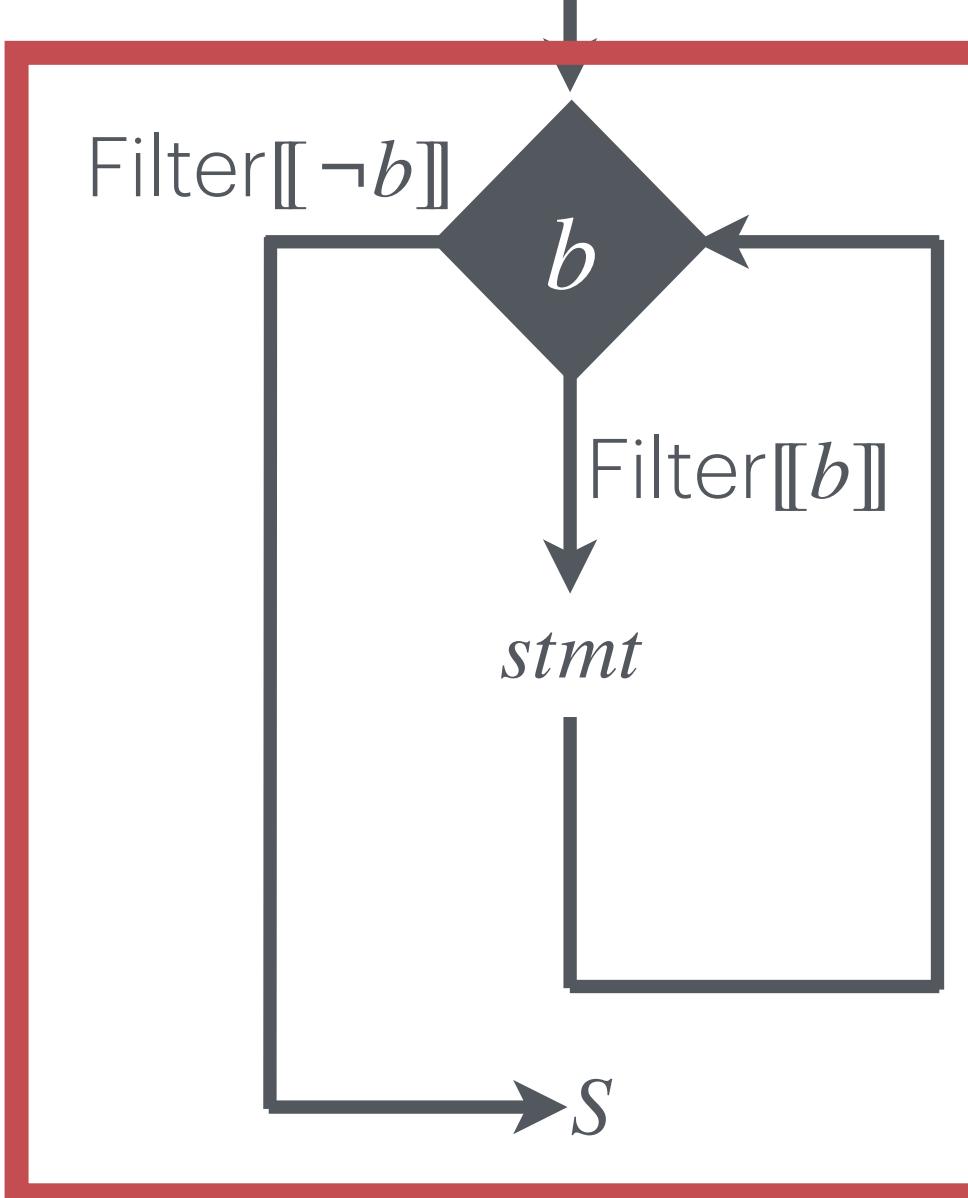
W = p

Modify the concrete semantics  
to collect the loop counter

$$\Lambda[\text{while } b \text{ do } stmt]S \triangleq \lim_n F^n$$

$$F(x) \triangleq \text{Filter}[\neg b]S \cup \boxed{\text{Filter}[b](\Lambda[stmt](x))}$$

# The RANGE Quantifier



p z m x n y

$$\text{Add}(3, z, 2, [4, 2], 2, [6, 8]) = z \mapsto [1, 1, 0]$$

$$\text{Add}(2, z, 2, [4, 2], 2, [6, 8]) = z \mapsto [1, 0]$$

$$\text{Add}(4, z, 2, [4, 2], 2, [6, 8]) = z \mapsto [0, 1, 1, 0]$$

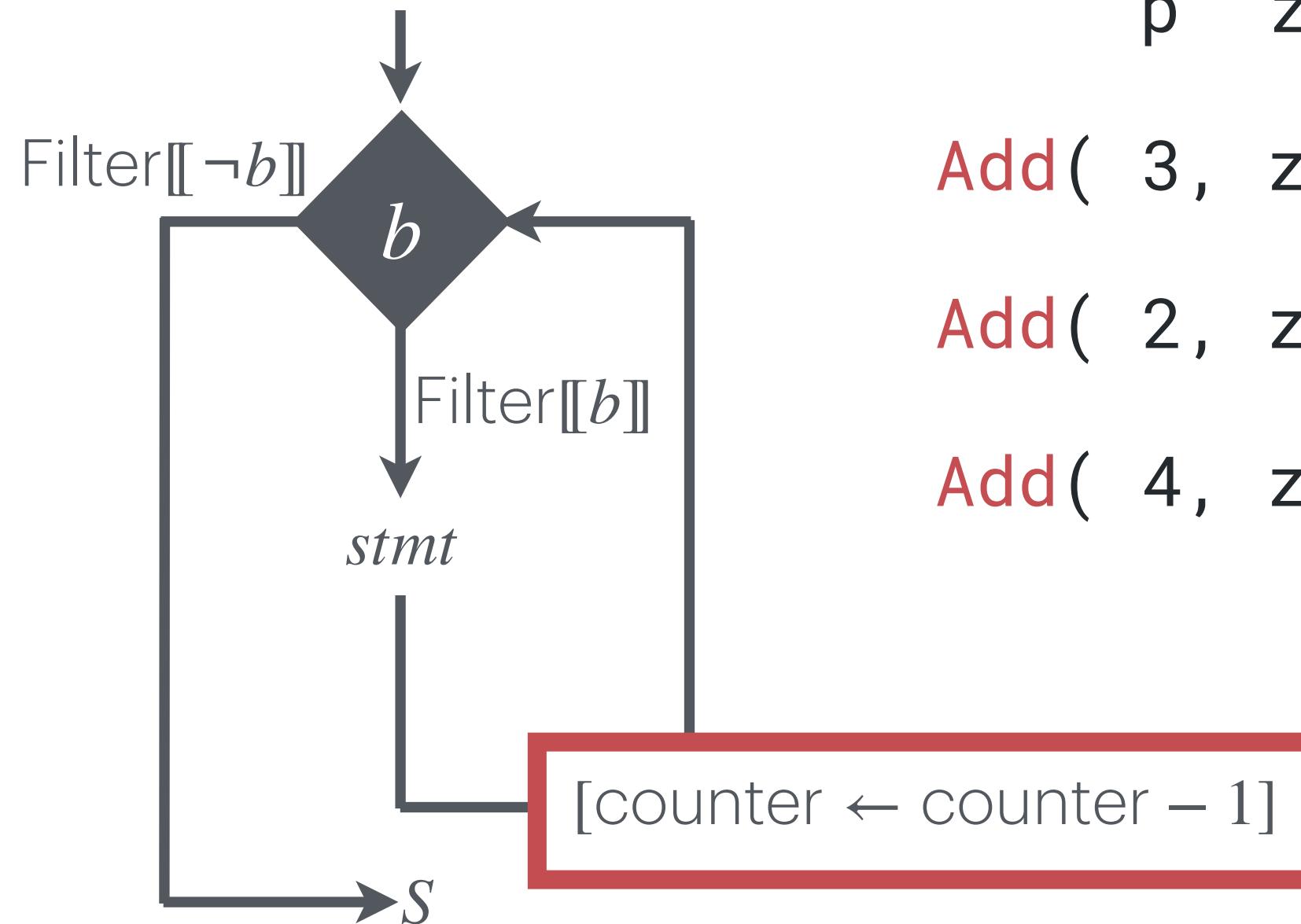
$W = p$

Modify the concrete semantics  
to **collect** the **loop counter**

$$\Lambda[\text{while } b \text{ do } stmt]S \triangleq \lim_n F^n$$

$$F(x) \triangleq \boxed{\text{Filter}[\neg b]S \cup \text{Filter}[b](\Lambda[stmt](x))}$$

# The RANGE Quantifier



p z m x n y

$$\text{Add}(3, z, 2, [4, 2], 2, [6, 8]) = z \mapsto [1, 1, 0]$$

$$\text{Add}(2, z, 2, [4, 2], 2, [6, 8]) = z \mapsto [1, 0]$$

$$\text{Add}(4, z, 2, [4, 2], 2, [6, 8]) = z \mapsto [0, 1, 1, 0]$$

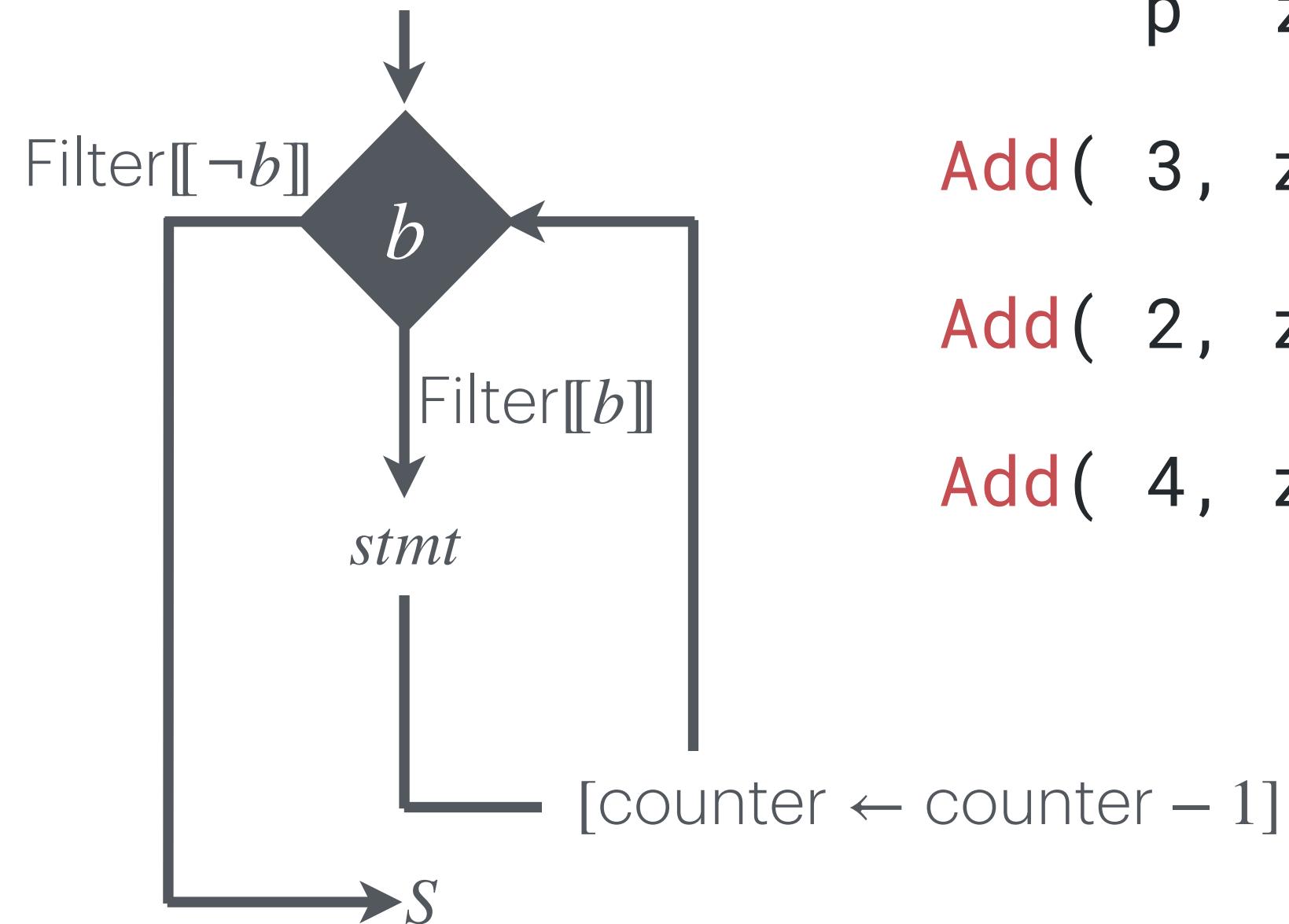
$W = p$

Modify the concrete semantics  
to **collect** the **loop counter**

$$\Lambda[\text{while } b \text{ do } stmt]S \triangleq \lim_n F^n$$

$$F(x) \triangleq \text{Filter}[\neg b]S \cup \text{Filter}[b](\Lambda[stmt](x[\text{counter} \leftarrow \text{counter} - 1]))$$

# The RANGE Quantifier



p z m x n y

$$\text{Add}(3, z, 2, [4, 2], 2, [6, 8]) = z \mapsto [1, 1, 0]$$

$$\text{Add}(2, z, 2, [4, 2], 2, [6, 8]) = z \mapsto [1, 0]$$

$$\text{Add}(4, z, 2, [4, 2], 2, [6, 8]) = z \mapsto [0, 1, 1, 0]$$

$W = p$

Modify the concrete semantics  
to **collect** the **loop counter**

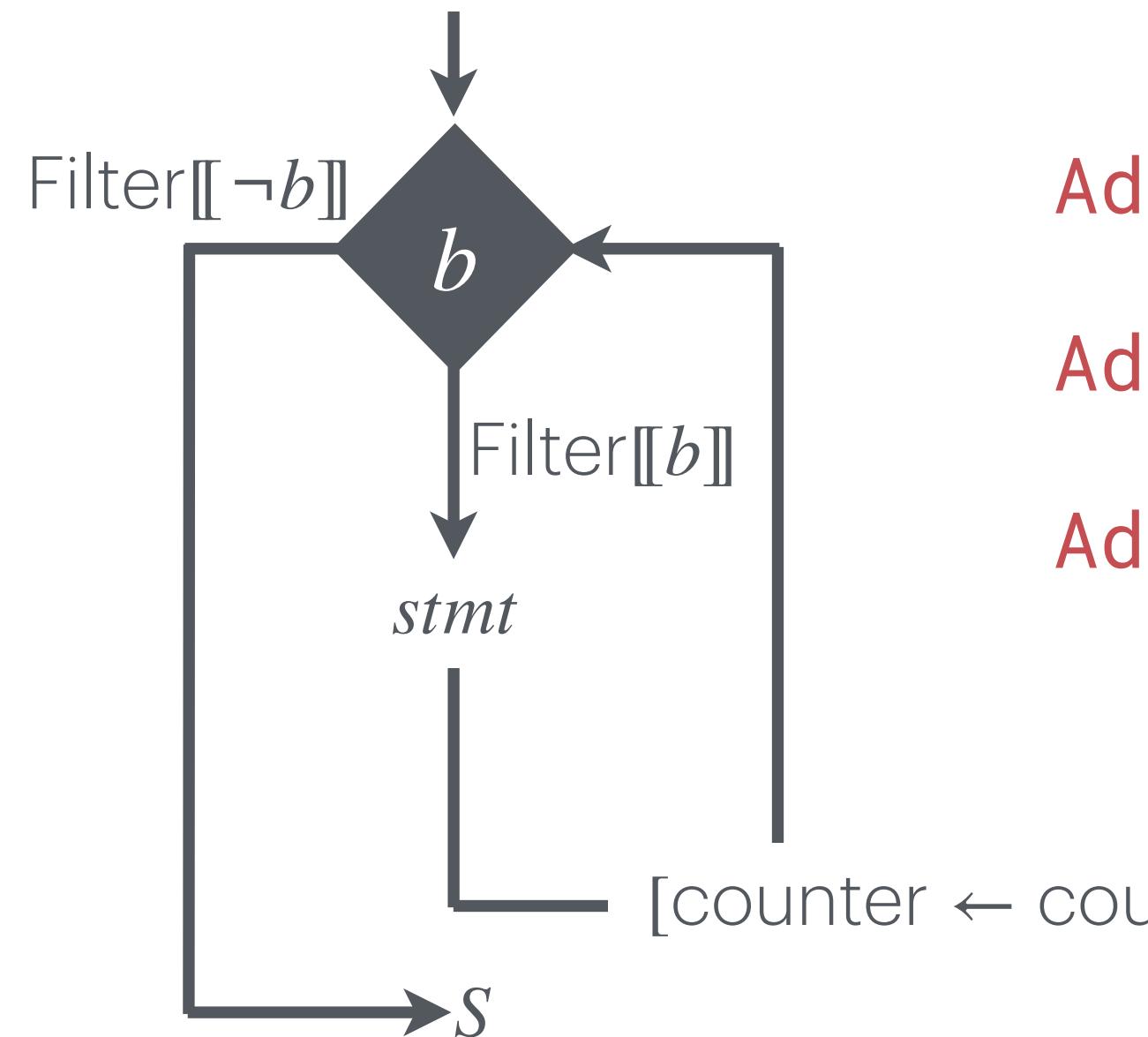
(Backwards) Entry point:

$$\Lambda[P]S \triangleq \Lambda[P](S[\text{counter} \leftarrow 0])$$

$$\Lambda[\text{while } b \text{ do } stmt]S \triangleq \lim_n F^n$$

$$F(x) \triangleq \text{Filter}[[\neg b]]S \cup \text{Filter}[[b]](\Lambda[stmt](x[\text{counter} \leftarrow \text{counter} - 1]))$$

# The RANGE Quantifier



	p	z	m	x	n	y	counter	
Add(	3,	z,	2,	[4,2],	2,	[6,8],	3 )	= z ↦ [1,1,0]
Add(	2,	z,	2,	[4,2],	2,	[6,8],	2 )	= z ↦ [1,0]
Add(	4,	z,	2,	[4,2],	2,	[6,8],	4 )	= z ↦ [0,1,1,0]

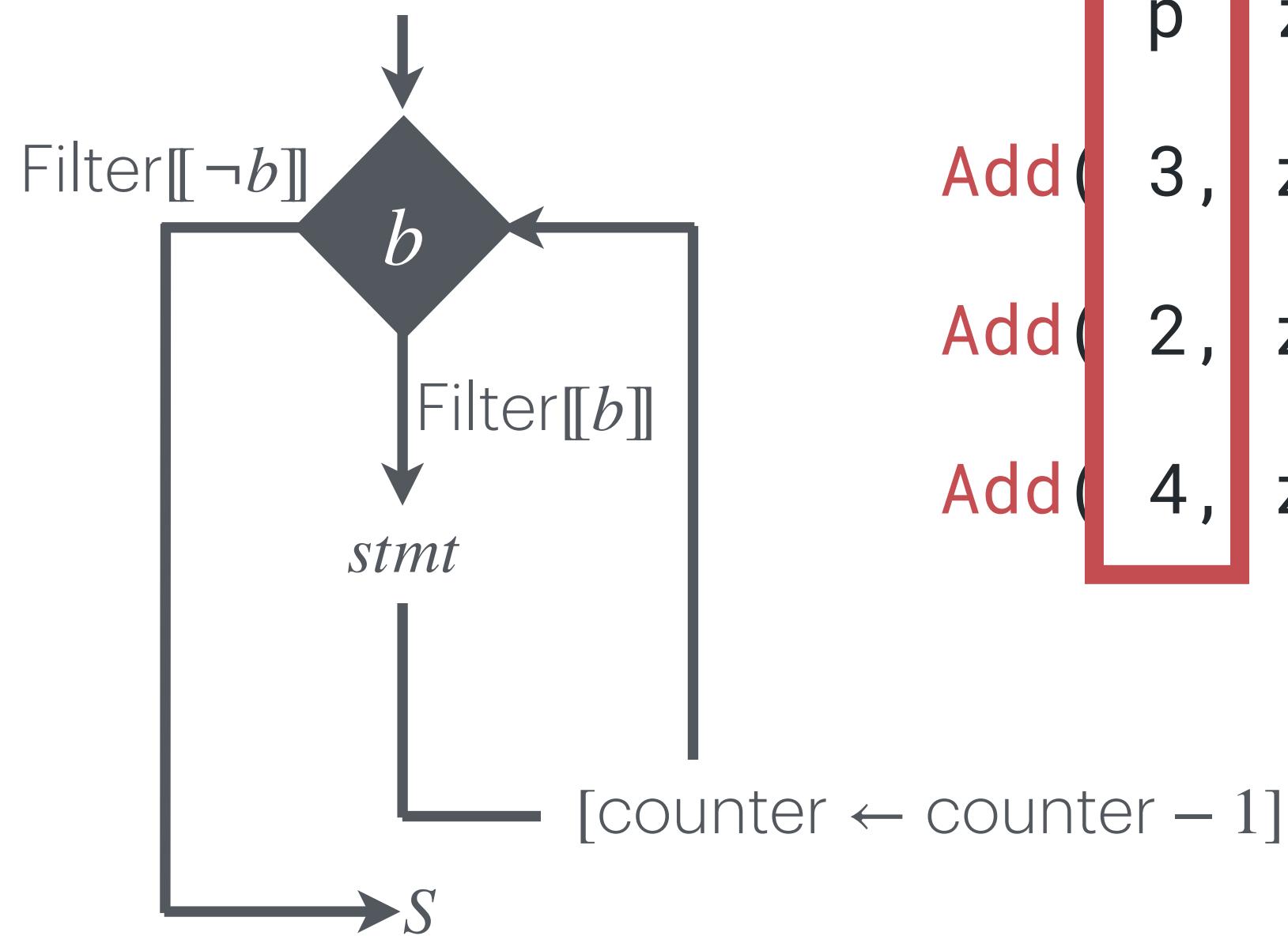
$W = p$

(Backwards) Entry point:  $\Lambda[P]S \triangleq \Lambda[P](S[\text{counter} \leftarrow 0])$

$$\Lambda[\text{while } b \text{ do } stmt]S \triangleq \lim_n F^n$$

$$F(x) \triangleq \text{Filter}[\neg b]S \cup \text{Filter}[b](\Lambda[stmt](x[\text{counter} \leftarrow \text{counter} - 1]))$$

# The RANGE Quantifier

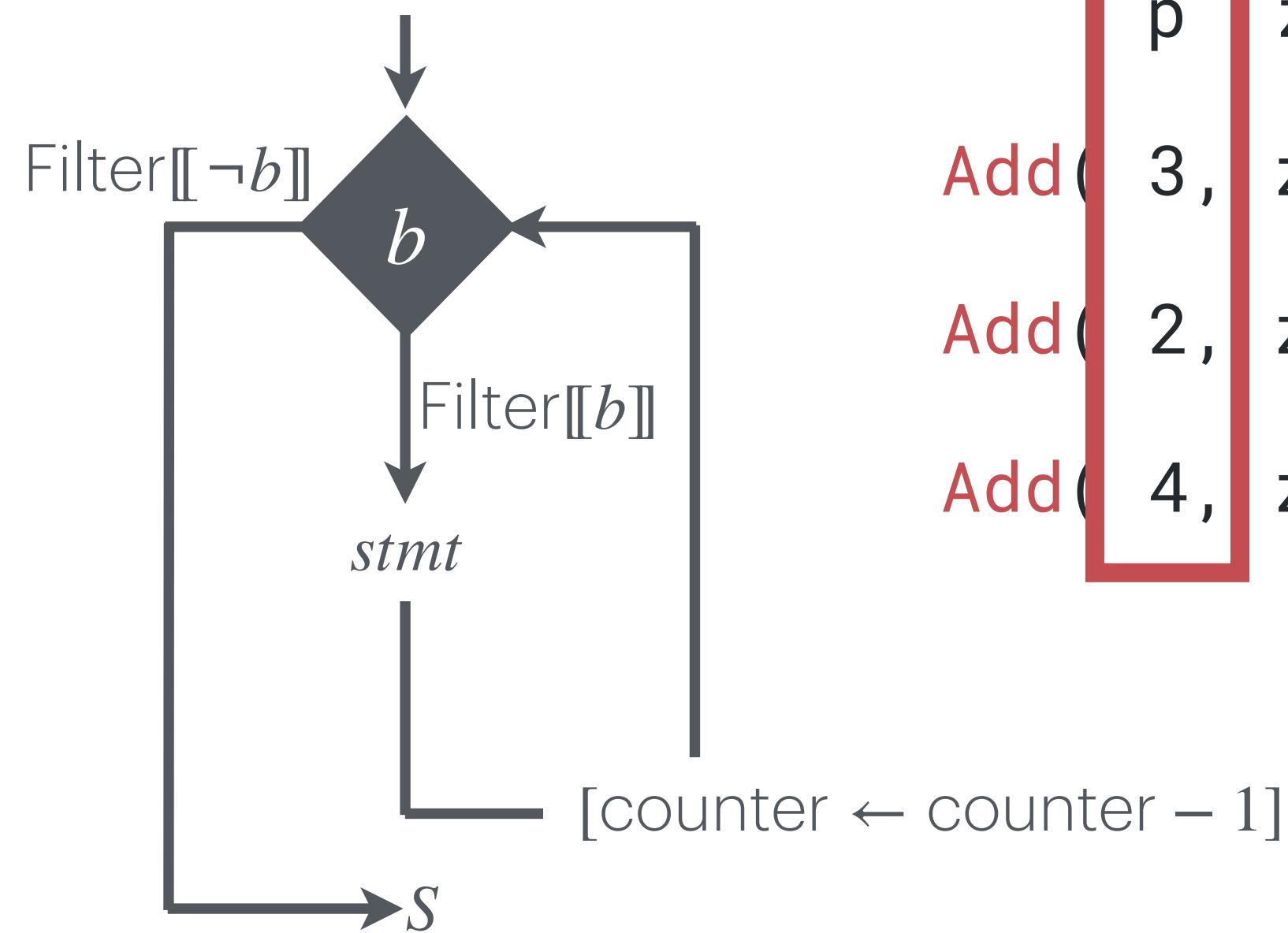


p	z	m	x	n	y	counter
Add( 3,	z,	2,	[4,2],	2,	[6,8],	3 ) = z ↦ [1,1,0]
Add( 2,	z,	2,	[4,2],	2,	[6,8],	2 ) = z ↦ [1,0]
Add( 4,	z,	2,	[4,2],	2,	[6,8],	4 ) = z ↦ [0,1,1,0]

$W = p$

RANGE quantifies changes wrt the **loop counter**

# The RANGE Quantifier



p	z	m	x	n	y	counter
Add( 3,	z,	2,	[4,2],	2,	[6,8],	3 ) = z ↦ [1,1,0]
Add( 2,	z,	2,	[4,2],	2,	[6,8],	2 ) = z ↦ [1,0]
Add( 4,	z,	2,	[4,2],	2,	[6,8],	4 ) = z ↦ [0,1,1,0]

$W = p$

Intensional

Extensional

RANGE quantifies changes wrt the **loop counter**

# Computing $\text{Range}_x^\natural(P)$

---

# Computing Range $_x^\natural(P)$

---

## 1. Remove Irrelevant Instructions

# Computing Range $_{\chi}^{\natural}(P)$

---

1. Remove Irrelevant Instructions

**Syntactic Dependency Analysis**

# Computing $\text{Range}_x^\natural(P)$

---

1. Remove Irrelevant Instructions

## Syntactic Dependency Analysis

2. Backward Abstract Analysis

# Computing Range $_x^\natural(P)$

---

1. Remove Irrelevant Instructions

**Syntactic Dependency Analysis**

2. Backward Abstract Analysis

**Invariant** on input variables + **counter**

# Computing Range $_{\chi}^{\natural}(P)$

---

1. Remove Irrelevant Instructions

**Syntactic Dependency Analysis**

2. Backward Abstract Analysis

**Invariant** on input variables + **counter**

3. Quantify the Impact

# Computing Range $_{\chi}^{\natural}(P)$

---

1. Remove Irrelevant Instructions

**Syntactic Dependency Analysis**

2. Backward Abstract Analysis

**Invariant** on input variables + **counter**

3. Quantify the Impact

**Mixed-integer linear programming**

# Computing $\text{Range}_x^\natural(P)$

---

1. Remove Irrelevant Instructions

**Syntactic Dependency Analysis**

2. Backward Abstract Analysis

**Invariant** on input variables + **counter**

3. Quantify the Impact

**Mixed-integer linear programming**

# 1. Irrelevant Instructions

```
1 def Add(p, z, m, x, n, y): 26
2     r = min(p, m) 27
3     s = min(p, n) 28
4     if (r < s): 29
5         t = p - s 30
6         q = s - r 31
7         i = 0 32
8         a = 0 33
9         for (; r > 0; r--): 34
10            s = x[i] 35
11            w = y[i] 36
12            z[i] = s + w + a 37
13            i = i + 1 38
14            a = (w < a) || 39
15            (s + w < s) || 40
16            (s + w + a < s) 41
17         do: 42
18            r = y[i] 43
19            b = (r < a) || 44
20            (r + a < r) 45
21            z[i] = r + a 46
22            i = i + 1 47
23            q-- 48
24            a = b 49
25        while (q > 0) 50
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
else:
    t = p - r
    q = r - s
    i = 0
    b = 0
    for (; s > 0; s--):
        r = x[i]
        w = y[i]
        z[i] = r + w + b
        i = i + 1
        b = (w < b) ||
             (r + w < r) ||
             (r + w + b < r)
    for (; q > 0; q--):
        r = x[i]
        z[i] = r + b
        i = i + 1
        b = (r < b) ||
             (r + b < r)
    if (t > 0):
        z[i] = b
        while (t > 0):
            i = i + 1
            t--
            if (t > 0):
                z[i] = 0
```

# 1. Irrelevant Instructions

```
1 def Add(p, z, m, x, n, y): 26
2     r = min(p, m) 27
3     s = min(p, n) 28
4     if (r < s): 29
5         t = p - s 30
6         q = s - r 31
7         i = 0 32
8         a = 0 33
9         for (; r > 0; r--): 34
10            s = x[i] 35
11            w = y[i] 36
12            z[i] = s + w + a 37
13            i = i + 1 38
14            a = (w < a) || 39
15            (s + w < s) || 40
16            (s + w + a < s) 41
17         do: 42
18            r = y[i] 43
19            b = (r < a) || 44
20            (r + a < r) 45
21            z[i] = r + a 46
22            i = i + 1 47
23            q-- 48
24            a = b 49
25        while (q > 0) 50
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
else:
    t = p - r
    q = r - s
    i = 0
    b = 0
    for (; s > 0; s--):
        r = x[i]
        w = y[i]
        z[i] = r + w + b
        i = i + 1
        b = (w < b) ||
             (r + w < r) ||
             (r + w + b < r)
    for (; q > 0; q--):
        r = x[i]
        z[i] = r + b
        i = i + 1
        b = (r < b) ||
             (r + b < r)
    if (t > 0):
        z[i] = b
        while (t > 0):
            i = i + 1
            t--
            if (t > 0):
                z[i] = 0
```

# 1. Irrelevant Instructions

```
1 def Add(p, z, m, x, n, y): 26
2     r = min(p, m) 27
3     s = min(p, n) 28
4     if (r < s): 29
5         t = p - s 30
6         q = s - r 31
7         i = 0 32
8         a = 0 33
9         for ( ; r > 0; r--): 34
10            s = x[i] 35
11            w = y[i] 36
12            z[i] = s + w + a 37
13            i = i + 1 38
14            a = (w < a) || 39
15                (s + w < s) || 40
16                (s + w + a < s) 41
17            do: 42
18                r = y[i] 43
19                b = (r < a) || 44
20                    (r + a < r) 45
21                z[i] = r + a 46
22                i = i + 1 47
23                q-- 48
24                a = b 49
25            while (q > 0) 50
26
27     else: 51
28         t = p - r
29         q = r - s
30         i = 0
31         b = 0
32         for ( ; s > 0; s--):
33             r = x[i]
34             w = y[i]
35             z[i] = r + w + b
36             i = i + 1
37             b = (w < b) ||
38                 (r + w < r) ||
39                 (r + w + b < r)
40         for ( ; q > 0; q--):
41             r = x[i]
42             z[i] = r + b
43             i = i + 1
44             b = (r < b) ||
45                 (r + b < r)
46         if (t > 0):
47             z[i] = b
48         while (t > 0):
49             i = i + 1
50             t--
51         if (t > 0):
52             z[i] = 0
```

# 1. Irrelevant Instructions

```
1 def Add(p, z, m, x, n, y): 26
2     r = min(p, m) 27
3     s = min(p, n) 28
4     if (r < s): 29
5         t = p - s 30
6         q = s - r 31
7         i = 0 32
8         a = 0 33
9         for ( ; r > 0; r--): 34
10            s = x[i] 35
11            w = y[i] 36
12            z[i] = s + w + a 37
13            i = i + 1 38
14            a = (w < a) || 39
15                (s + w < s) || 40
16                (s + w + a < s) 41
17            do: 42
18                r = y[i] 43
19                b = (r < a) || 44
20                    (r + a < r) 45
21                z[i] = r + a 46
22                i = i + 1 47
23                q-- 48
24                a = b 49
25            while (q > 0) 50
26
27     else: 51
28         t = p - r
29         q = r - s
30         i = 0
31         b = 0
32         for ( ; s > 0; s--):
33             r = x[i]
34             w = y[i]
35             z[i] = r + w + b
36             i = i + 1
37             b = (w < b) ||
38                 (r + w < r) ||
39                 (r + w + b < r)
40         for ( ; q > 0; q--):
41             r = x[i]
42             z[i] = r + b
43             i = i + 1
44             b = (r < b) ||
45                 (r + b < r)
46         if (t > 0):
47             z[i] = b
48         while (t > 0):
49             i = i + 1
50             t--
51         if (t > 0):
52             z[i] = 0
```

# 1. Irrelevant Instructions

```
1 def Add(p, z, m, x, n, y): 26
2     r = min(p, m)                27
3     s = min(p, n)                28
4     if (r < s):                 29
5         t = p - s                30
6         q = s - r                31
7         i = 0                     32
8         a = 0                     33
9         for (; r > 0; r--):      34
10        s = x[i]                  35
11        w = y[i]                  36
12        z[i] = s + w + a         37
13        i = i + 1                 38
14        a = (w < a) ||          39
15        (s + w < s) ||          40
16        (s + w + a < s)          41
17     do:                         42
18        r = y[i]                  43
19        b = (r < a) ||          44
20        (r + a < r)             45
21        z[i] = r + a              46
22        i = i + 1                 47
23        q--                      48
24        a = b                     49
25     while (q > 0)               50
26                                         51
27                                         52
28                                         53
29                                         54
30                                         55
31                                         56
32                                         57
33                                         58
34                                         59
35                                         60
36                                         61
37                                         62
38                                         63
39                                         64
40                                         65
41                                         66
42                                         67
43                                         68
44                                         69
45                                         70
46                                         71
47                                         72
48                                         73
49                                         74
50                                         75
51                                         76
52                                         77
53                                         78
54                                         79
55                                         80
56                                         81
57                                         82
58                                         83
59                                         84
59                                         85
60                                         86
61                                         87
62                                         88
63                                         89
64                                         90
65                                         91
66                                         92
67                                         93
68                                         94
69                                         95
70                                         96
71                                         97
72                                         98
73                                         99
74                                         100
75                                         101
76                                         102
77                                         103
78                                         104
79                                         105
80                                         106
81                                         107
82                                         108
83                                         109
84                                         110
85                                         111
86                                         112
87                                         113
88                                         114
89                                         115
90                                         116
91                                         117
92                                         118
93                                         119
94                                         120
95                                         121
96                                         122
97                                         123
98                                         124
99                                         125
100                                         126
101                                         127
102                                         128
103                                         129
104                                         130
105                                         131
106                                         132
107                                         133
108                                         134
109                                         135
110                                         136
111                                         137
112                                         138
113                                         139
114                                         140
115                                         141
116                                         142
117                                         143
118                                         144
119                                         145
120                                         146
121                                         147
122                                         148
123                                         149
124                                         150
125                                         151
126                                         152
127                                         153
128                                         154
129                                         155
130                                         156
131                                         157
132                                         158
133                                         159
134                                         160
135                                         161
136                                         162
137                                         163
138                                         164
139                                         165
140                                         166
141                                         167
142                                         168
143                                         169
144                                         170
145                                         171
146                                         172
147                                         173
148                                         174
149                                         175
150                                         176
151                                         177
152                                         178
153                                         179
154                                         180
155                                         181
156                                         182
157                                         183
158                                         184
159                                         185
160                                         186
161                                         187
162                                         188
163                                         189
164                                         190
165                                         191
166                                         192
167                                         193
168                                         194
169                                         195
170                                         196
171                                         197
172                                         198
173                                         199
174                                         200
175                                         201
176                                         202
177                                         203
178                                         204
179                                         205
180                                         206
181                                         207
182                                         208
183                                         209
184                                         210
185                                         211
186                                         212
187                                         213
188                                         214
189                                         215
190                                         216
191                                         217
192                                         218
193                                         219
194                                         220
195                                         221
196                                         222
197                                         223
198                                         224
199                                         225
200                                         226
201                                         227
202                                         228
203                                         229
204                                         230
205                                         231
206                                         232
207                                         233
208                                         234
209                                         235
210                                         236
211                                         237
212                                         238
213                                         239
214                                         240
215                                         241
216                                         242
217                                         243
218                                         244
219                                         245
220                                         246
221                                         247
222                                         248
223                                         249
224                                         250
225                                         251
226                                         252
227                                         253
228                                         254
229                                         255
230                                         256
231                                         257
232                                         258
233                                         259
234                                         260
235                                         261
236                                         262
237                                         263
238                                         264
239                                         265
240                                         266
241                                         267
242                                         268
243                                         269
244                                         270
245                                         271
246                                         272
247                                         273
248                                         274
249                                         275
250                                         276
251                                         277
252                                         278
253                                         279
254                                         280
255                                         281
256                                         282
257                                         283
258                                         284
259                                         285
260                                         286
261                                         287
262                                         288
263                                         289
264                                         290
265                                         291
266                                         292
267                                         293
268                                         294
269                                         295
270                                         296
271                                         297
272                                         298
273                                         299
274                                         300
275                                         301
276                                         302
277                                         303
278                                         304
279                                         305
280                                         306
281                                         307
282                                         308
283                                         309
284                                         310
285                                         311
286                                         312
287                                         313
288                                         314
289                                         315
290                                         316
291                                         317
292                                         318
293                                         319
294                                         320
295                                         321
296                                         322
297                                         323
298                                         324
299                                         325
299                                         326
300                                         327
301                                         328
302                                         329
303                                         330
304                                         331
305                                         332
306                                         333
307                                         334
308                                         335
309                                         336
310                                         337
311                                         338
312                                         339
313                                         340
314                                         341
315                                         342
316                                         343
317                                         344
318                                         345
319                                         346
320                                         347
321                                         348
322                                         349
323                                         350
324                                         351
325                                         352
326                                         353
327                                         354
328                                         355
329                                         356
330                                         357
331                                         358
332                                         359
333                                         360
334                                         361
335                                         362
336                                         363
337                                         364
338                                         365
339                                         366
340                                         367
341                                         368
342                                         369
343                                         370
344                                         371
345                                         372
346                                         373
347                                         374
348                                         375
349                                         376
350                                         377
351                                         378
352                                         379
353                                         380
354                                         381
355                                         382
356                                         383
357                                         384
358                                         385
359                                         386
360                                         387
361                                         388
362                                         389
363                                         390
364                                         391
365                                         392
366                                         393
367                                         394
368                                         395
369                                         396
370                                         397
371                                         398
372                                         399
373                                         400
374                                         401
375                                         402
376                                         403
377                                         404
378                                         405
379                                         406
380                                         407
381                                         408
382                                         409
383                                         410
384                                         411
385                                         412
386                                         413
387                                         414
388                                         415
389                                         416
390                                         417
391                                         418
392                                         419
393                                         420
394                                         421
395                                         422
396                                         423
397                                         424
398                                         425
399                                         426
399                                         427
400                                         428
401                                         429
402                                         430
403                                         431
404                                         432
405                                         433
406                                         434
407                                         435
408                                         436
409                                         437
410                                         438
411                                         439
412                                         440
413                                         441
414                                         442
415                                         443
416                                         444
417                                         445
418                                         446
419                                         447
420                                         448
421                                         449
422                                         450
423                                         451
424                                         452
425                                         453
426                                         454
427                                         455
428                                         456
429                                         457
430                                         458
431                                         459
432                                         460
433                                         461
434                                         462
435                                         463
436                                         464
437                                         465
438                                         466
439                                         467
440                                         468
441                                         469
442                                         470
443                                         471
444                                         472
445                                         473
446                                         474
447                                         475
448                                         476
449                                         477
450                                         478
451                                         479
452                                         480
453                                         481
454                                         482
455                                         483
456                                         484
457                                         485
458                                         486
459                                         487
460                                         488
461                                         489
462                                         490
463                                         491
464                                         492
465                                         493
466                                         494
467                                         495
468                                         496
469                                         497
470                                         498
471                                         499
472                                         500
473                                         501
474                                         502
475                                         503
476                                         504
477                                         505
478                                         506
479                                         507
480                                         508
481                                         509
482                                         510
483                                         511
484                                         512
485                                         513
486                                         514
487                                         515
488                                         516
489                                         517
490                                         518
491                                         519
492                                         520
493                                         521
494                                         522
495                                         523
496                                         524
497                                         525
498                                         526
499                                         527
499                                         528
500                                         529
501                                         530
502                                         531
503                                         532
504                                         533
505                                         534
506                                         535
507                                         536
508                                         537
509                                         538
510                                         539
511                                         540
512                                         541
513                                         542
514                                         543
515                                         544
516                                         545
517                                         546
518                                         547
519                                         548
520                                         549
521                                         550
522                                         551
523                                         552
524                                         553
525                                         554
526                                         555
527                                         556
528                                         557
529                                         558
530                                         559
531                                         560
532                                         561
533                                         562
534                                         563
535                                         564
536                                         565
537                                         566
538                                         567
539                                         568
540                                         569
541                                         570
542                                         571
543                                         572
544                                         573
545                                         574
546                                         575
547                                         576
548                                         577
549                                         578
550                                         579
551                                         580
552                                         581
553                                         582
554                                         583
555                                         584
556                                         585
557                                         586
558                                         587
559                                         588
560                                         589
561                                         590
562                                         591
563                                         592
564                                         593
565                                         594
566                                         595
567                                         596
568                                         597
569                                         598
570                                         599
571                                         600
572                                         601
573                                         602
574                                         603
575                                         604
576                                         605
577                                         606
578                                         607
579                                         608
580                                         609
581                                         610
582                                         611
583                                         612
584                                         613
585                                         614
586                                         615
587                                         616
588                                         617
589                                         618
590                                         619
591                                         620
592                                         621
593                                         622
594                                         623
595                                         624
596                                         625
597                                         626
598                                         627
599                                         628
599                                         629
600                                         630
601                                         631
602                                         632
603                                         633
604                                         634
605                                         635
606                                         636
607                                         637
608                                         638
609                                         639
610                                         640
611                                         641
612                                         642
613                                         643
614                                         644
615                                         645
616                                         646
617                                         647
618                                         648
619                                         649
620                                         650
621                                         651
622                                         652
623                                         653
624                                         654
625                                         655
626                                         656
627                                         657
628                                         658
629                                         659
630                                         660
631                                         661
632                                         662
633                                         663
634                                         664
635                                         665
636                                         666
637                                         667
638                                         668
639                                         669
640                                         670
641                                         671
642                                         672
643                                         673
644                                         674
645                                         675
646                                         676
647                                         677
648                                         678
649                                         679
650                                         680
651                                         681
652                                         682
653                                         683
654                                         684
655                                         685
656                                         686
657                                         687
658                                         688
659                                         689
660                                         690
661                                         691
662                                         692
663                                         693
664                                         694
665                                         695
666                                         696
667                                         697
668                                         698
669                                         699
670                                         700
671                                         701
672                                         702
673                                         703
674                                         704
675                                         705
676                                         706
677                                         707
678                                         708
679                                         709
680                                         710
681                                         711
682                                         712
683                                         713
684                                         714
685                                         715
686                                         716
687                                         717
688                                         718
689                                         719
690                                         720
691                                         721
692                                         722
693                                         723
694                                         724
695                                         725
696                                         726
697                                         727
698                                         728
699                                         729
699                                         730
700                                         731
701                                         732
702                                         733
703                                         734
704                                         735
705                                         736
706                                         737
707                                         738
708                                         739
709                                         740
710                                         741
711                                         742
712                                         743
713                                         744
714                                         745
715                                         746
716                                         747
717                                         748
718                                         749
719                                         750
720                                         751
721                                         752
722                                         753
723                                         754
724                                         755
725                                         756
726                                         757
727                                         758
728                                         759
729                                         760
730                                         761
731                                         762
732                                         763
733                                         764
734                                         765
735                                         766
736                                         767
737                                         768
738                                         769
739                                         770
740                                         771
741                                         772
742                                         773
743                                         774
744                                         775
745                                         776
746                                         777
747                                         778
748                                         779
749                                         780
750                                         781
751                                         782
752                                         783
753                                         784
754                                         785
755                                         786
756                                         787
757                                         788
758                                         789
759                                         790
760                                         791
761                                         792
762                                         793
763                                         794
764                                         795
765                                         796
766                                         797
767                                         798
768                                         799
769                                         800
770                                         801
771                                         802
772                                         803
773                                         804
774                                         805
775                                         806
776                                         807
777                                         808
778                                         809
779                                         810
780                                         811
781                                         812
782                                         813
783                                         814
784                                         815
785                                         816
786                                         817
787                                         818
788                                         819
789                                         820
790                                         821
791                                         822
792                                         823
793                                         824
794                                         825
795                                         826
796                                         827
797                                         828
798                                         829
799                                         830
799                                         831
800                                         832
801                                         833
802                                         834
803                                         835
804                                         836
805                                         837
806                                         838
807                                         839
808                                         840
809                                         841
810                                         842
811                                         843
812                                         844
813                                         845
814                                         846
815                                         847
816                                         848
817                                         849
818                                         850
819                                         851
820                                         852
821                                         853
822                                         854
823                                         855
824                                         856
825                                         857
826                                         858
827                                         859
828                                         860
829                                         861
830                                         862
831                                         863
832                                         864
833                                         865
834                                         866
835                                         867
836                                         868
837                                         869
838                                         870
839                                         871
840                                         872
841                                         873
842                                         874
843                                         875
844                                         876
845                                         877
846                                         878
847                                         879
8
```

# 1. Irrelevant Instructions

```
1 def Add(p, z, m, x, n, y): 26
2     r = min(p, m) 27
3     s = min(p, n) 28
4     if (r < s): 29
5         t = p - s 30
6         q = s - r 31
7         i = 0 32
8         a = 0 33
9         for (; r > 0; r--): 34
10            s = x[i] 35
11            w = y[i] 36
12            z[i] = s + w + a 37
13            i = i + 1 38
14            a = (w < a) || 39
15            (s + w < s) || 40
16            (s + w + a < s) 41
17         do: 42
18            r = y[i] 43
19            b = (r < a) || 44
20            (r + a < r) 45
21            z[i] = r + a 46
22            i = i + 1 47
23            q-- 48
24            a = b 49
25        while (q > 0) 50
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
else:
    t = p - r
    q = r - s
    i = 0
    b = 0
    for (; s > 0; s--):
        r = x[i]
        w = y[i]
        z[i] = r + w + b
        i = i + 1
        b = (w < b) ||
             (r + w < r) ||
             (r + w + b < r)
    for (; q > 0; q--):
        r = x[i]
        z[i] = r + b
        i = i + 1
        b = (r < b) ||
             (r + b < r)
    if (t > 0):
        z[i] = b
        while (t > 0):
            i = i + 1
            t--
    if (t > 0):
        z[i] = 0
```

# 1. Irrelevant Instructions

```

1 def Add(p, z, m, x, n, y): 26
2     r = min(p, m) 27
3     s = min(p, n) 28
4     if (r < s): 29
5         t = p - s 30
6         q = s - r 31
7         i = 0 32
8         a = 0 33
9         for (; r > 0; r--): 34
10            s = x[i] 35
11            w = y[i] 36
12            z[i] = s + w + a 37
13            i = i + 1 38
14            a = (w < a) || 39
15            (s + w < s) || 40
16            (s + w + a < s) 41
17         do: 42
18             r = y[i] 43
19             b = (r < a) || 44
20             (r + a < r) 45
21             z[i] = r + a 46
22             i = i + 1 47
23             q-- 48
24             a = b 49
25         while (q > 0) 50
26
27     else: 51
28         t = p - r
29         q = r - s
30         i = 0
31         b = 0
32         for (; s > 0; s--):
33             r = x[i]
34             w = y[i]
35             z[i] = r + w + b
36             i = i + 1
37             b = (w < b) ||
38             (r + w < r) ||
39             (r + w + b < r)
40         for (; q > 0; q--):
41             r = x[i]
42             z[i] = r + b
43             i = i + 1
44             b = (r < b) ||
45             (r + b < r)
46         if (t > 0):
47             z[i] = b
48             while (t > 0):
49                 i = i + 1
50                 t--
51             if (t > 0):
52                 z[i] = 0

```

## 1. Irrelevant Instructions

# Syntactic Dependency Analysis

```
1 def Add(p, z, m, x, n, y): 26
2     r = min(p, m)                27
3     s = min(p, n)                28
4     if (r < s):                 29
5         t = p - s                30
6         q = s - r                31
7         i = 0                     32
8         a = 0                     33
9         for (; r > 0; r--):      34
10        s = x[i]                  35
11        w = y[i]                  36
12        z[i] = s + w + a         37
13        i = i + 1                 38
14        a = (w < a) ||          39
15        (s + w < s) ||          40
16        (s + w + a < s)          41
17    do:                         42
18        r = y[i]                  43
19        b = (r < a) ||          44
20        (r + a < r)             45
21        z[i] = r + a              46
22        i = i + 1                 47
23        q--                      48
24        a = b                     49
25    while (q > 0)               50
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
else:
    t = p - r
    q = r - s
    i = 0
    b = 0
    for (; s > 0; s--):
        r = x[i]
        w = y[i]
        z[i] = r + w + b
        i = i + 1
        b = (w < b) ||
            (r + w < r) ||
            (r + w + b < r)
    for (; q > 0; q--):
        r = x[i]
        z[i] = r + b
        i = i + 1
        b = (r < b) ||
            (r + b < r)
    if (t > 0):
        z[i] = b
        while (t > 0):
            i = i + 1
            t--
            if (t > 0):
                z[i] = 0
```

# 1. Irrelevant Instructions

---

## Syntactic Dependency Analysis

```
1  def Add(p, z, m, x, n, y):  
2      r = min(p, m)  
3      s = min(p, n)  
4      if (r < s):  
5          t = p - s  
6          q = s - r  
9          for (; r > 0; r--):  
--          T skip;  
17         do:  
23             I q--;  
25             while (q > 0)  
26         else:  
27             t = p - r  
28             q = r - s  
31             for (; s > 0; s--):  
--             T skip;  
39             for (; q > 0; q--):  
--             T skip;  
45             if (t > 0):  
47                 while (t > 0):  
49                     T t--;
```

# Computing $\text{Range}_x^\natural(P)$

---

1. Remove Irrelevant Instructions

**Syntactic Dependency Analysis**

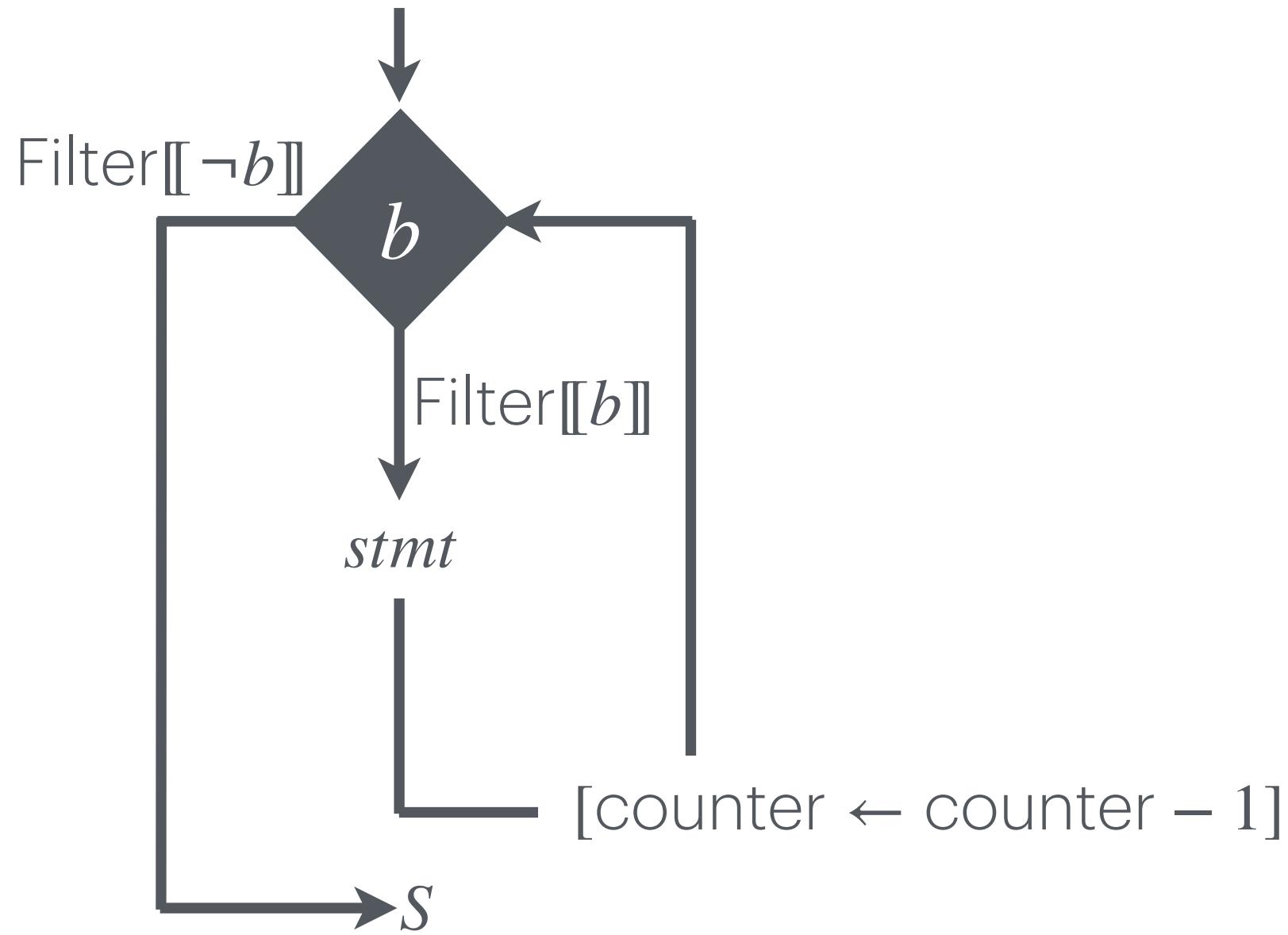
2. Backward Abstract Analysis

**Invariant** on input variables + **counter**

3. Quantify the Impact

**Mixed-integer linear programming**

## 2. Backward Abstract Analysis



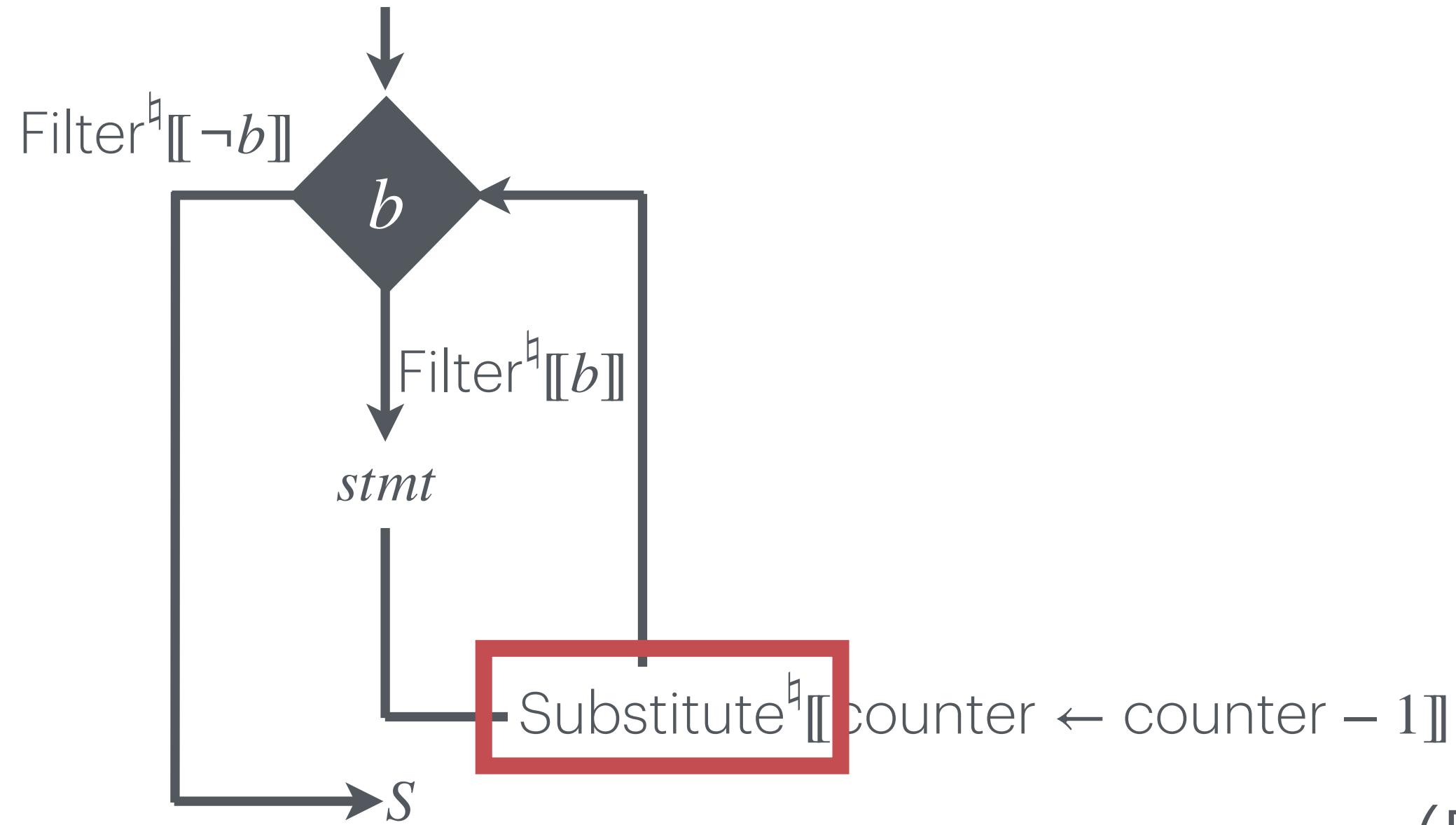
(Backwards) Entry point

$$\Lambda[P]S \triangleq \Lambda[P](S[\text{counter} \leftarrow 0])$$

$$\Lambda[\text{while } b \text{ do } stmt]S \triangleq \lim_n F^n$$

$$F(x) \triangleq \text{Filter}[\neg b]S \cup \text{Filter}[b](\Lambda[stmt](x[\text{counter} \leftarrow \text{counter} - 1]))$$

## 2. Backward Abstract Analysis



Abstract counterpart of  
concrete operators

(Backwards) Entry point

$$\Lambda^\natural[P]S^\natural \triangleq \Lambda^\natural[P](\text{Substitute}^\natural[\text{counter} \leftarrow 0]S^\natural)$$

$$\Lambda^\natural[\text{while } b \text{ do } stmt]S^\natural \triangleq \lim_n F^n$$

$$F(x^\natural) \triangleq \text{Filter}^\natural[\neg b]S \cup \text{Filter}^\natural[b](\Lambda^\natural[stmt](\text{Substitute}^\natural[\text{counter} \leftarrow \text{counter} - 1]x^\natural))$$

# Computing $\text{Range}_x^\natural(P)$

---

1. Remove Irrelevant Instructions

**Syntactic Dependency Analysis**

2. Backward Abstract Analysis

**Invariant** on input variables + **counter**

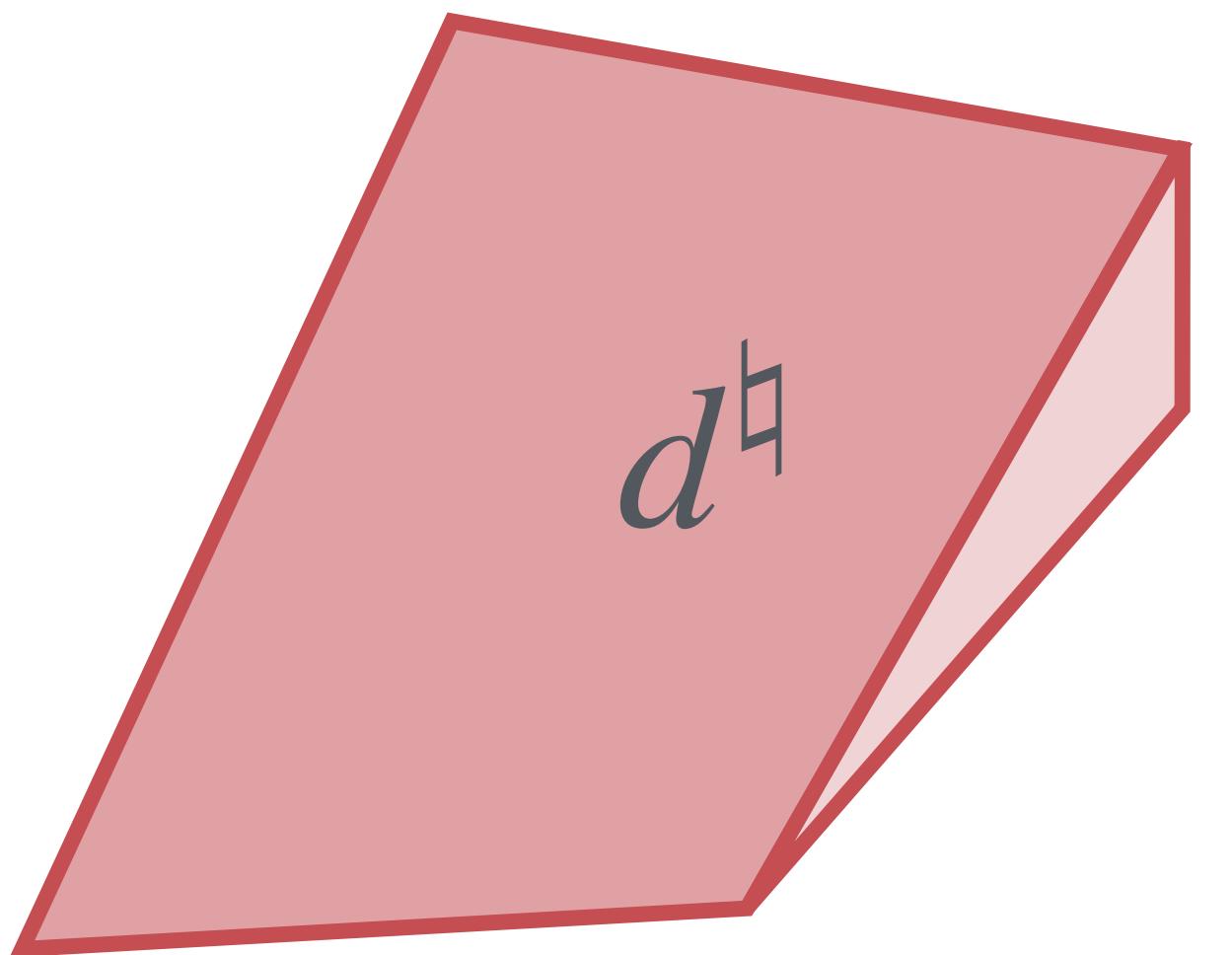
3. Quantify the Impact

**Mixed-integer linear programming**

# 3. Impact Quantification

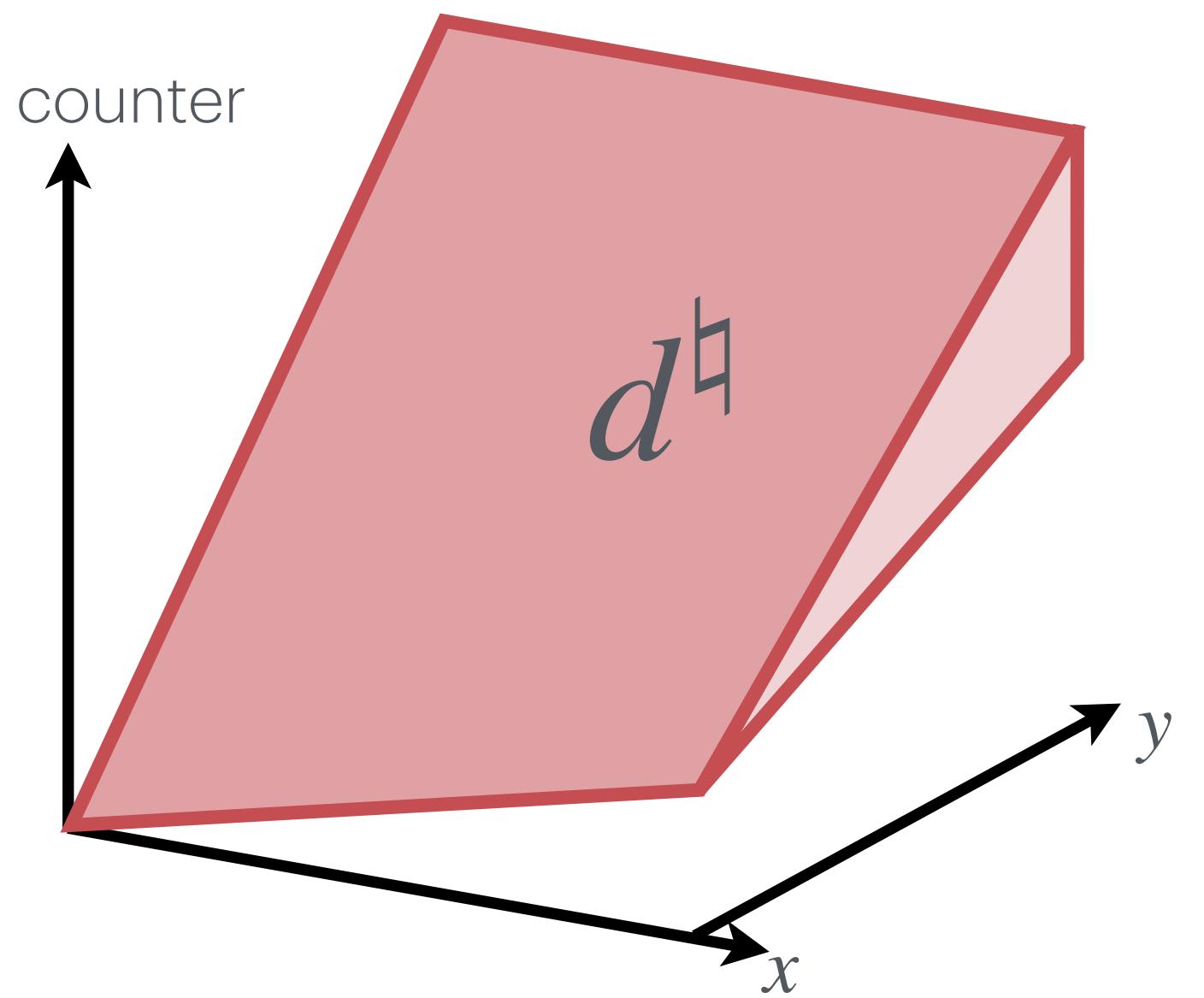
---

Abstract invariant on the  
**input variables + counter**



# 3. Impact Quantification

Abstract invariant on the  
**input variables + counter**

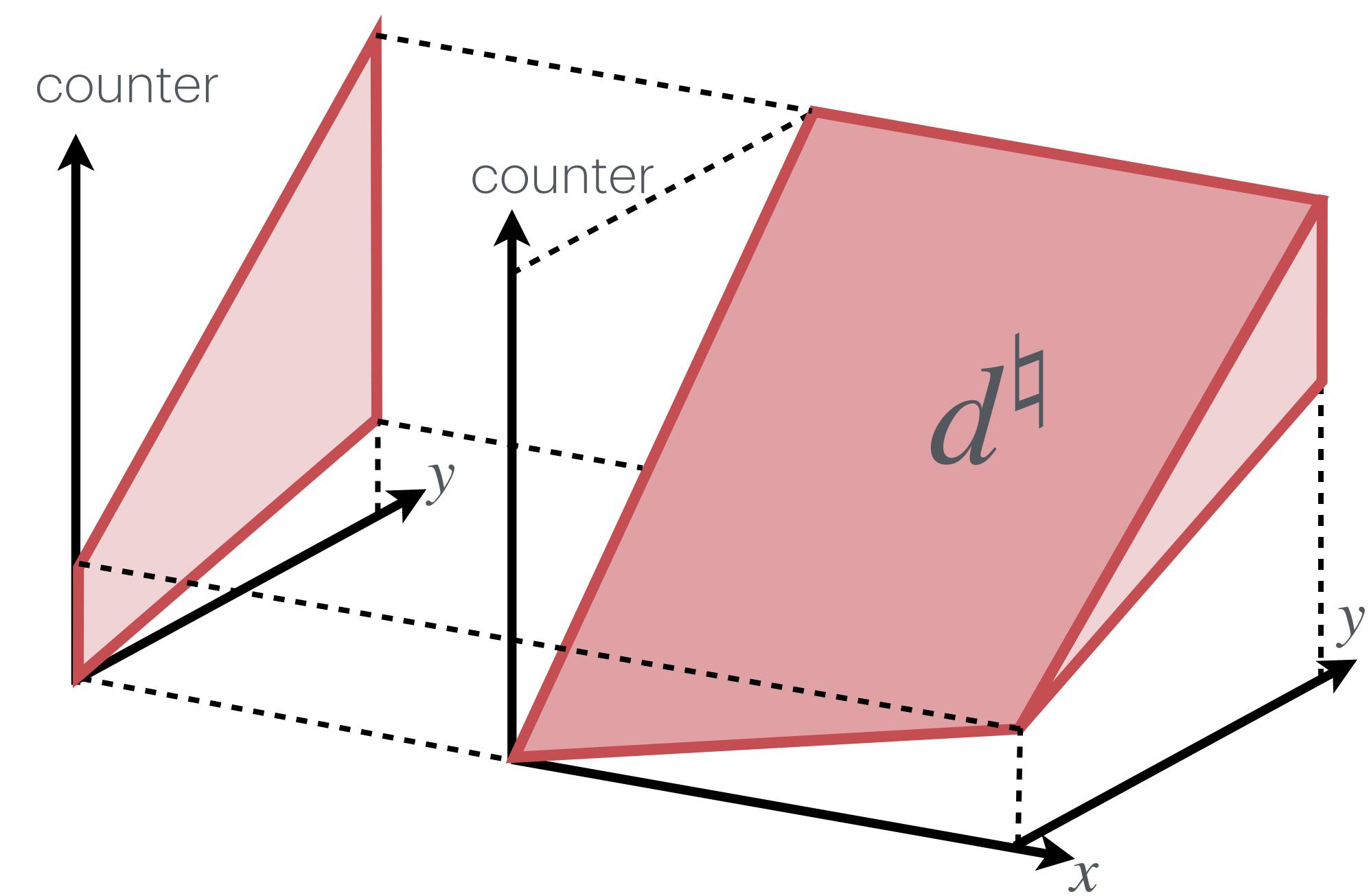


**$x$  input variables of interest**

# 3. Impact Quantification

Abstract invariant on the  
**input variables + counter**

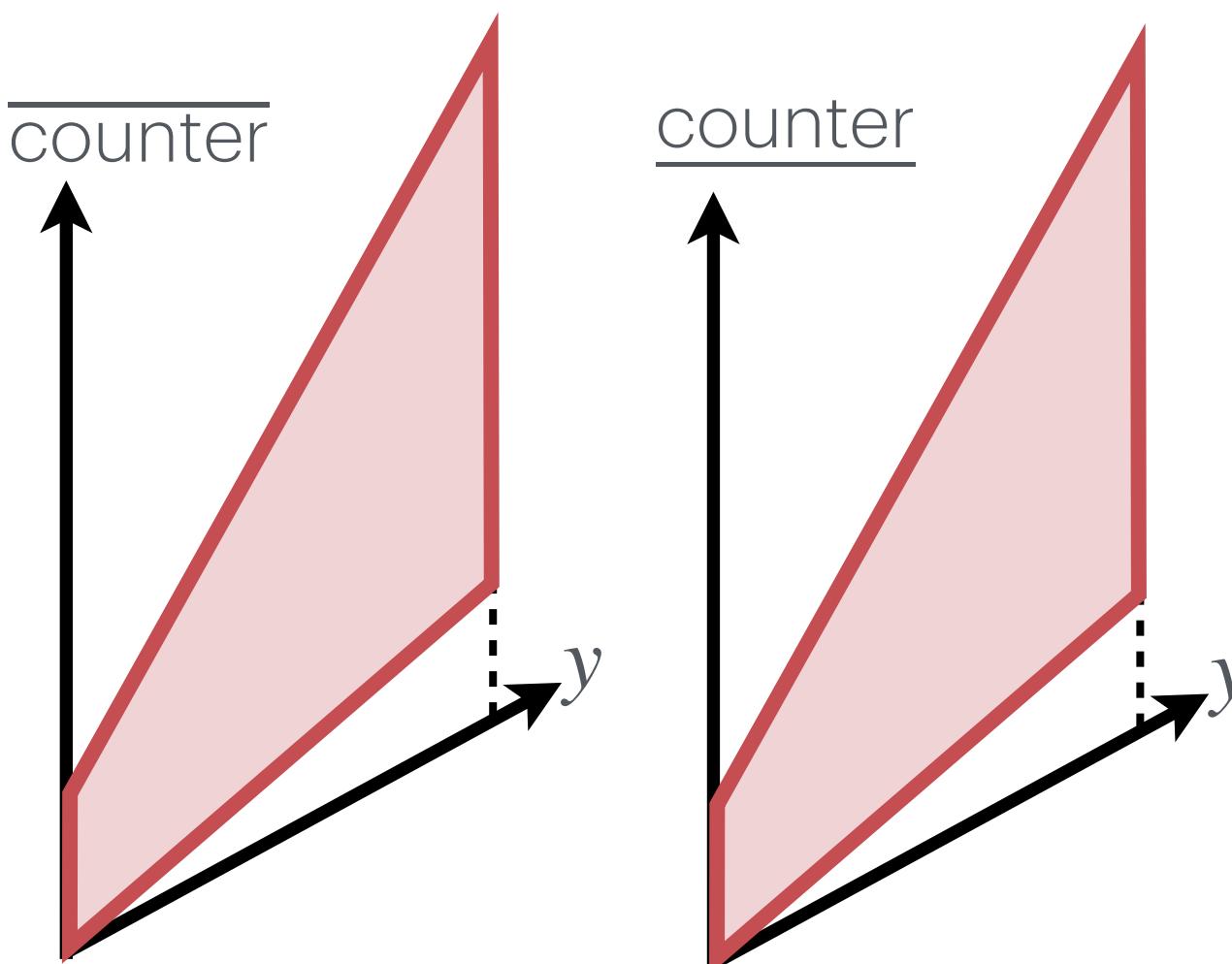
(1) **Project away  $x$**



**$x$  input variables of interest**

# 3. Impact Quantification

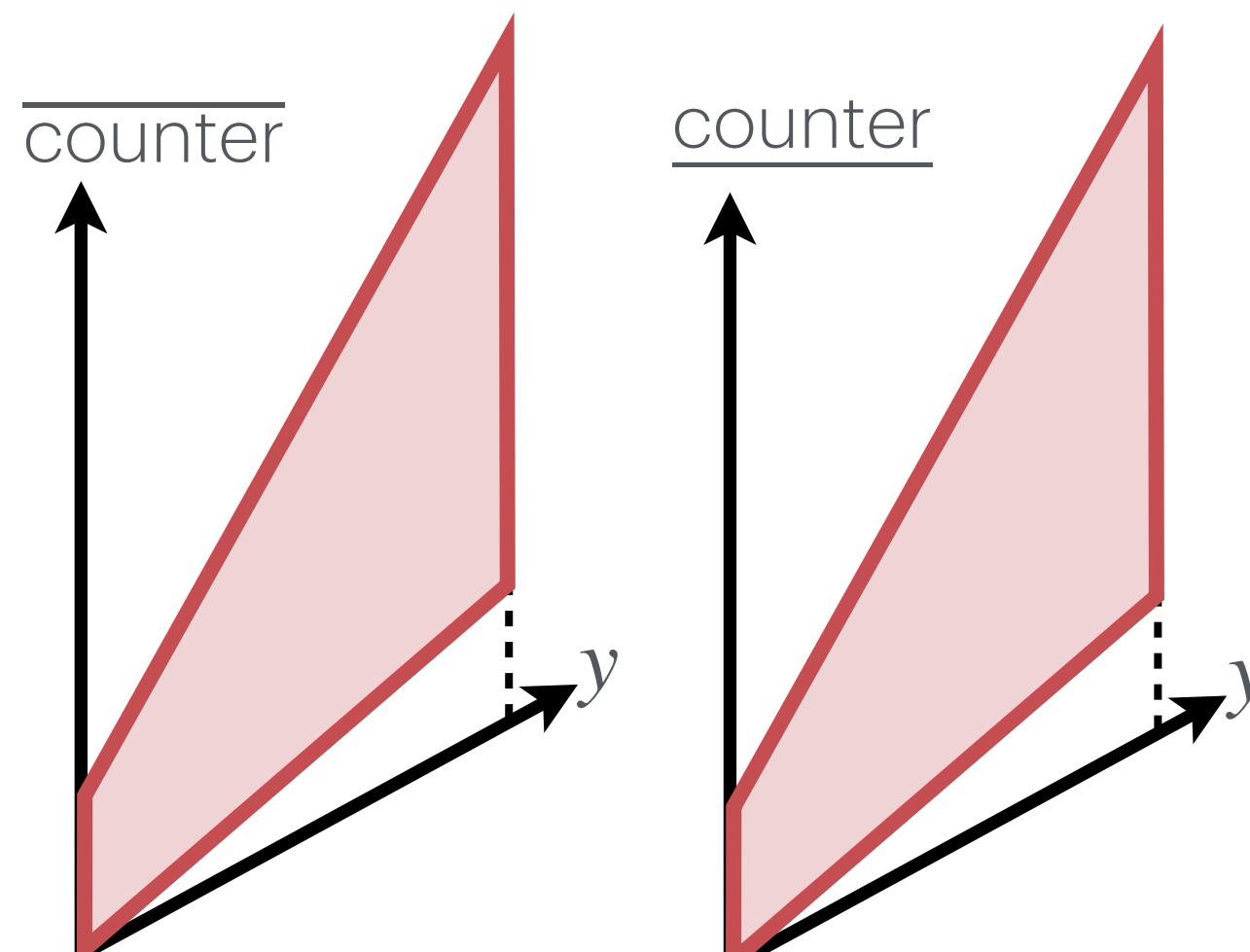
Abstract invariant on the  
**input variables + counter**



- (1) **Project** away  $x$
- (2) **Duplicate** the invariant and **substitute**  
the counter with counter and counter

# 3. Impact Quantification

Abstract invariant on the  
**input variables + counter**

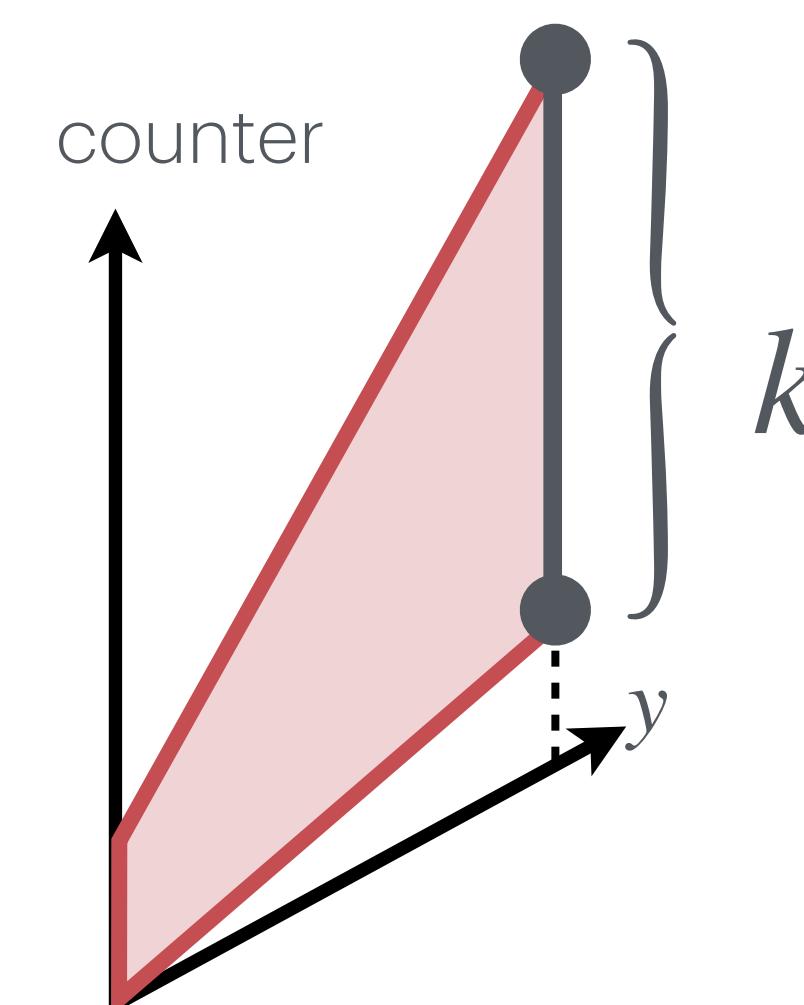


$$0 \leq k \leq \overline{\text{counter}} - \underline{\text{counter}}$$

- (1) **Project** away  $x$
- (2) **Duplicate** the invariant and **substitute**  
the counter with counter and counter
- (3) **Maximize** the distance between the two

# 3. Impact Quantification

Abstract invariant on the  
**input variables + counter**



$$0 \leq k \leq \overline{\text{counter}} - \underline{\text{counter}}$$

- (1) **Project** away  $x$
- (2) **Duplicate** the invariant and **substitute**  
the counter with counter and counter
- (3) **Maximize** the distance between the two

$k$  is the impact of  $x$

# Maximization Problem

---

$\text{Range}_x^\natural(d^\natural) = \max k$  subject to

Substitute $[\![\text{counter} \leftarrow \overline{\text{counter}}]\!]$ (Project $_x(d^\natural)$ )  $\wedge$

Substitute $[\![\text{counter} \leftarrow \underline{\text{counter}}]\!]$ (Project $_x(d^\natural)$ )  $\wedge$

$0 \leq k \leq \overline{\text{counter}} - \underline{\text{counter}}$

# Maximization Problem

---

$\text{Range}_x^\natural(d^\natural) = \max k$  subject to

(1) **Project** away  $x$

Substitute $\llbracket \text{counter} \leftarrow \overline{\text{counter}} \rrbracket (\text{Project}_x(d^\natural)) \wedge$

Substitute $\llbracket \text{counter} \leftarrow \underline{\text{counter}} \rrbracket (\text{Project}_x(d^\natural)) \wedge$

$0 \leq k \leq \overline{\text{counter}} - \underline{\text{counter}}$

# Maximization Problem

---

$\text{Range}_x^\natural(d^\natural) = \max k$  subject to

Substitute $[\![\text{counter} \leftarrow \overline{\text{counter}}]\!]$ (Project $_x(d^\natural)$ )  $\wedge$

Substitute $[\![\text{counter} \leftarrow \underline{\text{counter}}]\!]$ (Project $_x(d^\natural)$ )  $\wedge$

$0 \leq k \leq \overline{\text{counter}} - \underline{\text{counter}}$

(2) Duplication

# Maximization Problem

---

(3) **Maximize** the distance  $k$

$\text{Range}_x^\natural(d^\natural) = \boxed{\max k}$  subject to

Substitute $[\![\text{counter} \leftarrow \overline{\text{counter}}]\!]$ (Project $_x(d^\natural)$ )  $\wedge$

Substitute $[\![\text{counter} \leftarrow \underline{\text{counter}}]\!]$ (Project $_x(d^\natural)$ )  $\wedge$

$0 \leq k \leq \overline{\text{counter}} - \underline{\text{counter}}$

# Add Function

Forward +  
Backward abstract  
analysis ↪

```
1 def Add(p, z, m, x, n, y):  
2     r = min(p, m)  
3     s = min(p, n)  
4     if (r < s):  
5         t = p - s  
6         q = s - r  
9         for (; r > 0; r--):  
--             skip; counter--  
17         do:  
23             q--; counter--  
25             while (q > 0)  
26         else:  
27             t = p - r  
28             q = r - s  
31             for (; s > 0; s--):  
--                 skip; counter--  
39             for (; q > 0; q--):  
--                 skip; counter--  
45             if (t > 0):  
47                 while (t > 0):  
--                     t--; counter--  
49             assert counter = 0  
--
```

$d^\natural$  is  
 $p = \text{counter} \wedge 0 \leq p \leq u$

# Add Function

$d^\natural$  is

$p = \text{counter} \wedge 0 \leq p \leq u$

$\text{Range}_x^\natural(d^\natural) = \max k$  subject to

$\text{Substitute}[\text{counter} \leftarrow \overline{\text{counter}}](\text{Project}_x(d^\natural)) \wedge$

$\text{Substitute}[\text{counter} \leftarrow \underline{\text{counter}}](\text{Project}_x(d^\natural)) \wedge$

$0 \leq k \leq \overline{\text{counter}} - \underline{\text{counter}}$

# Add Function

$d^\natural$  is

$$p = \text{counter} \wedge 0 \leq p \leq u$$

Range $_{x^\natural}(d^\natural) = \max k$  subject to

Substitute $[\![\text{counter} \leftarrow \overline{\text{counter}}]\!]$ (Project $_{x^\natural}(d^\natural) \wedge$

Substitute $[\![\text{counter} \leftarrow \underline{\text{counter}}]\!]$ (Project $_{x^\natural}(d^\natural) \wedge$

$0 \leq k \leq \overline{\text{counter}} - \underline{\text{counter}}$

# Add Function

$d^\natural$  is

$p = \text{counter} \wedge 0 \leq p \leq u$

Range $^\natural(p = \text{counter} \wedge 0 \leq p \leq u) =$

max  $k$  subject to

$$0 \leq \overline{\text{counter}} \leq u \wedge$$

$$0 \leq \underline{\text{counter}} \leq u \wedge$$

$$0 \leq k \leq \overline{\text{counter}} - \underline{\text{counter}}$$

# Add Function

$d^\natural$  is

$p = \text{counter} \wedge 0 \leq p \leq u$

Range $^\natural(p = \text{counter} \wedge 0 \leq p \leq u) =$

max  $k$  subject to

$$\boxed{0 \leq \overline{\text{counter}} \leq u \wedge}$$
$$0 \leq \underline{\text{counter}} \leq u \wedge$$

$$0 \leq k \leq \overline{\text{counter}} - \underline{\text{counter}}$$

# Add Function

$d^\natural$  is

$p = \text{counter} \wedge 0 \leq p \leq u$

Range $^\natural(p = \text{counter} \wedge 0 \leq p \leq u) =$

max  $k$  subject to

$$0 \leq \overline{\text{counter}} \leq u \wedge$$

$$0 \leq \underline{\text{counter}} \leq u \wedge$$

$$0 \leq k \leq \overline{\text{counter}} - \underline{\text{counter}}$$

# Add Function

$d^\natural$  is

$$p = \text{counter} \wedge 0 \leq p \leq u$$

$$\text{Range}_p^\natural(p = \text{counter} \wedge 0 \leq p \leq u) =$$

max  $k$  subject to

$$0 \leq \overline{\text{counter}} \leq u \wedge$$

$$0 \leq \underline{\text{counter}} \leq u \wedge$$

$$0 \leq k \leq \overline{\text{counter}} - \underline{\text{counter}}$$

$u$

# Add Function

$d^\natural$  is

$p = \text{counter} \wedge 0 \leq p \leq u$

Range $_{\frac{n}{n}}^\natural(p = \text{counter} \wedge 0 \leq p \leq u) =$

max  $k$  subject to

$p = \overline{\text{counter}} \wedge$

$p = \underline{\text{counter}} \wedge$

$0 \leq p \leq u \wedge$

$0 \leq k \leq \overline{\text{counter}} - \underline{\text{counter}}$

# Add Function

$d^\natural$  is

$p = \text{counter} \wedge 0 \leq p \leq u$

Range $_{\frac{n}{n}}^\natural(p = \text{counter} \wedge 0 \leq p \leq u) =$

max  $k$  subject to

$$\left. \begin{array}{l} p = \overline{\text{counter}} \wedge \\ p = \underline{\text{counter}} \wedge \\ 0 \leq p \leq u \wedge \\ 0 \leq k \leq \overline{\text{counter}} - \underline{\text{counter}} \end{array} \right\} 0$$

# Add Function

$d^\natural$  is

$p = \text{counter} \wedge 0 \leq p \leq u$

$$\text{Range}_n^\natural(p = \text{counter} \wedge 0 \leq p \leq u) = \left\{ \begin{array}{l} \max k \text{ subject to} \\ \boxed{\begin{array}{l} p = \overline{\text{counter}} \wedge \\ p = \underline{\text{counter}} \wedge \\ 0 \leq p \leq u \wedge \\ \boxed{0 \leq k \leq \overline{\text{counter}} - \underline{\text{counter}}} \end{array}} \end{array} \right\} 0$$

# Add Function

$$\text{Range}_p(d^\natural) = u$$

```
1 def Add(p, z, m, x, n, y):  
2     r = min(p, m)  
3     s = min(p, n)  
4     if (r < s):  
5         t = p - s  
6         q = s - r  
9         for (; r > 0; r--):  
--             T skip; counter--  
17         do:  
23             I q--; counter--  
25             while (q > 0)  
26         else:  
27             t = p - r  
28             q = r - s  
31             for (; s > 0; s--):  
--                 T skip; counter--  
39             for (; q > 0; q--):  
--                 T skip; counter--  
45             if (t > 0):  
47                 while (t > 0):  
49                     T t--; counter--  
--             assert counter = 0
```

$d^\natural$  is  
 $p = \text{counter} \wedge 0 \leq p \leq u$

# Add Function

$$\text{Range}_p^{\natural}(d^{\natural}) = u$$

$$\text{Range}_n^{\natural}(d^{\natural}) = 0$$

$$\text{Range}_m^{\natural}(d^{\natural}) = 0$$

$$\text{Range}_z^{\natural}(d^{\natural}) = 0$$

$$\text{Range}_x^{\natural}(d^{\natural}) = 0$$

$$\text{Range}_y^{\natural}(d^{\natural}) = 0$$

```
1 def Add(p, z, m, x, n, y):  
2     r = min(p, m)  
3     s = min(p, n)  
4     if (r < s):  
5         t = p - s  
6         q = s - r  
9         for (; r > 0; r--):  
10            T skip; counter--  
17        do:  
18            I q--; counter--  
23            while (q > 0)  
25        else:  
26            t = p - r  
27            q = r - s  
28            for (; s > 0; s--):  
29                T skip; counter--  
39            for (; q > 0; q--):  
40                T skip; counter--  
45            if (t > 0):  
47                while (t > 0):  
48                    T t--; counter--  
49            assert counter = 0  
--
```

$d^{\natural}$  is  
 $p = \text{counter} \wedge 0 \leq p \leq u$

# Add Function

$$\text{Range}_p^{\natural}(d^{\natural}) = u$$

$$\text{Range}_n^{\natural}(d^{\natural}) = 0$$

$$\text{Range}_m^{\natural}(d^{\natural}) = 0$$

$$\text{Range}_z^{\natural}(d^{\natural}) = 0$$

$$\text{Range}_x^{\natural}(d^{\natural}) = 0$$

$$\text{Range}_y^{\natural}(d^{\natural}) = 0$$

```
1 def Add(p, z, m, x, n, y):
2     r = min(p, m)
3     s = min(p, n)
4     if (r < s):
5         t = p - s
6         q = s - r
7         for (; r > 0; r--):
8             T skip; counter--
9             do:
10                 I q--; counter--
11                 while (q > 0)
12                     else:
13                         t = p - r
14                         q = r - s
15                         for (; s > 0; s--):
16                             T skip; counter--
17                             for (; q > 0; q--):
18                                 T skip; counter--
19                                 if (t > 0):
20                                     while (t > 0):
21                                         T t--; counter--
22                                         assert counter = 0
23                                         --
```

$d^{\natural}$  is  
 $p = \text{counter} \wedge 0 \leq p \leq u$

The **Add** function is **safe**  
from **timing side-channels**

on **input variables**

$m, n, x, y, z$

[github.com/denismazzucato/timesec](https://github.com/denismazzucato/timesec)

Python + Abstract Domain Library Apron

github.com/denismazzucato/**timesec**

Python + Abstract Domain Library Apron

## S2N-Bignum AWS

<https://github.com/awslabs/s2n-bignum>

- used in cryptographic applications
- 72 disassembled c routines, 5984 loc
- 1172 variables (272 input variables)

github.com/denismazzucato/timesec

Python + Abstract Domain Library Apron

## S2N-Bignum AWS

<https://github.com/awslabs/s2n-bignum>

- used in cryptographic applications
- 72 disassembled c routines, 5984 loc
- 1172 variables (272 input variables)

## & SVComp

Software  
Verification  
Competition

# S2N-Bignum AWS



Verified that the S2N-Bignum library is **timing side-channel safe** for **data** input variables

PROGRAM	INPUT VARIABLES $\Delta$	MAYBE DANGEROUS	ZERO IMPACT
	SAFE $\Delta _S$	NUMERICAL $\Delta _N$	
Add	$s_1, s_3, s_5$	$n_2, n_4, n_6$	$s_1$ $s_3, s_5, n_2, n_4, n_6$
Amontifier	$s_1$	$n_2, n_3, n_4$	$s_1$ $n_2, n_3, n_4$
Amontmul	$s_1$	$n_2, n_3, n_4, n_5$	$s_1$ $n_2, n_3, n_4, n_5$
Amontredc	$s_1, s_3, s_6$	$n_2, n_4, n_5$	$s_1, s_3, s_6$ $n_2, n_4, n_5$
Amontsqr	$s_1$	$n_2, n_3, n_4$	$s_1$ $n_2, n_3, n_4$
Bitfield	$s_1$	$n_2, n_3, n_4, n_5$	$s_1$ $n_2, n_3, n_4, n_5$
Bitsize	$s_1$	$n_2$	$s_1$ $n_2$
Cdiv	$s_1, s_3$	$n_2, n_4, n_5$	$s_1, s_3$ $n_2, n_4, n_5$
Cdiv_exact	$s_1, s_3$	$n_2, n_4, n_5$	$s_1$ $n_2, s_3, n_4, n_5$
Cld	$s_1$	$n_2$	$s_1$ $n_2$
Clz	$s_1$	$n_2$	$s_1$ $n_2$
Cmadd	$s_1, s_4$	$n_2, n_3, n_5$	$s_1, s_4$ $n_2, n_3, n_5$
Cmnegadd	$s_1, s_4$	$n_2, n_3, n_5$	$s_1, s_4$ $n_2, n_3, n_5$
Cmod	$s_1$	$n_2, n_3$	$s_1$ $n_2, n_3$
Cmul	$s_1, s_4$	$n_2, n_3, n_5$	$s_1, s_4$ $n_2, n_3, n_5$
Cprime	$s_1, s_3$	$n_2, n_4, n_5$	$s_1, s_3$ $n_2, n_4, n_5$
Copy	$s_1, s_3$	$n_2, n_4$	$s_1, s_3$ $n_2, n_4$
Copy_row_from_table	$s_3, s_4$	$n_1, n_2, n_5$	$s_3, s_4$ $n_1, n_2, n_5$
Copy_row_from_table_16_neon	$s_3$	$n_1, n_2, n_4$	$s_3$ $n_1, n_2, n_4$
Copy_row_from_table_32_neon	$s_3$	$n_1, n_2, n_4$	$s_3$ $n_1, n_2, n_4$
Copy_row_from_table_8n_neon	$s_3, s_4$	$n_1, n_2, n_5$	$s_3, s_4$ $n_1, n_2, n_5$
Ctd	$s_1$	$n_2$	$s_1$ $n_2$
Ctz	$s_1$	$n_2$	$s_1$ $n_2$
Demon	$s_1$	$n_2, n_3, n_4$	$s_1$ $n_2, n_3, n_4$
Digit	$s_1$	$n_2, n_3$	$s_1$ $n_2, n_3$
Digitsize	$s_1$	$n_2$	$s_1$ $n_2$
Divmod10	$s_1$	$n_2$	$s_1$ $n_2$
Emontredc	$s_1$	$n_2, n_3, n_4$	$s_1$ $n_2, n_3, n_4$
Eq	$s_1, s_3$	$n_2, n_4$	$s_1, s_3$ $n_2, n_4$
Even	$s_1$	$n_2$	$s_1, s_2$
Ge	$s_1, s_3$	$n_2, n_4$	$s_1, s_3$ $n_2, n_4$
Gt	$s_1, s_3$	$n_2, n_4$	$s_1, s_3$ $n_2, n_4$
Iszero	$s_1$	$n_2$	$s_1$ $n_2$
Le	$s_1, s_3$	$n_2, n_4$	$s_1, s_3$ $n_2, n_4$
Lt	$s_1, s_3$	$n_2, n_4$	$s_1, s_3$ $n_2, n_4$
Madd	$s_1, s_3, s_5$	$n_2, n_4, n_6$	$s_1, s_3, s_5$ $n_2, n_3, n_4, n_5$
Modadd	$s_1$	$n_2, n_3, n_4, n_5$	$s_1$ $n_2, n_3, n_4, n_5$
Moddouble	$s_1$	$n_2, n_3, n_4$	$s_1$ $n_2, n_3, n_4$
Modifier	$s_1$	$n_2, n_3, n_4$	$s_1$ $n_2, n_3, n_4$
Modinv	$s_1$	$n_2, n_3, n_4, n_5$	$s_1$ $n_2, n_3, n_4, n_5$
Modoptneg	$s_1$	$n_2, n_3, n_4, n_5$	$s_1$ $n_2, n_3, n_4, n_5$
Modsub	$s_1$	$n_2, n_3, n_4, n_5$	$s_1$ $n_2, n_3, n_4, n_5$
Montifier	$s_1$	$n_2, n_3, n_4$	$s_1$ $n_2, n_3, n_4$
Montmul	$s_1$	$n_2, n_3, n_4, n_5$	$s_1$ $n_2, n_3, n_4, n_5$
Montredc	$s_1, s_3, s_6$	$n_2, n_4, n_5$	$s_1, s_3, s_6$ $n_2, n_4, n_5$
Montsqr	$s_1$	$n_2, n_3, n_4$	$s_1$ $n_2, n_3, n_4$
Mul	$s_1, s_3, s_5$	$n_2, n_4, n_6$	$s_1, s_3, s_5$ $n_2, n_4, n_6$
Muladd10	$s_1$	$n_2, n_3$	$s_1$ $n_2, n_3$
Mux	$s_2$	$n_1, n_3, n_4, n_5$	$s_2$ $n_1, n_3, n_4, n_5$
Mux16	$s_1$	$n_2, n_3, n_4$	$s_1$ $n_2, n_3, n_4$
Negmodinv	$s_1$	$n_2, n_3$	$s_1$ $n_2, n_3$
Nonzero	$s_1$	$n_2$	$s_1$ $n_2$
Normalize	$s_1$	$n_2$	$s_1$ $n_2$
Odd	$s_1$	$n_2$	$s_1$ $n_2$
Of_word	$s_1$	$n_2, n_3$	$s_1$ $n_2, n_3$
Optadd	$s_1$	$n_2, n_3, n_4, n_5$	$s_1$ $n_2, n_3, n_4, n_5$
Optneg	$s_1$	$n_2, n_3, n_4$	$s_1$ $n_2, n_3, n_4$
Optsub	$s_1$	$n_2, n_3, n_4, n_5$	$s_1$ $n_2, n_3, n_4, n_5$
Optsubadd	$s_1$	$n_2, n_3, n_4, n_5$	$s_1$ $n_2, n_3, n_4, n_5$
Pow2	$s_1$	$n_2, n_3$	$s_1$ $n_2, n_3$
Shl_small	$s_1, s_3$	$n_2, n_4, n_5$	$s_1, s_3$ $n_2, n_4, n_5$
Shr_small	$s_1, s_3$	$n_2, n_4, n_5$	$s_1, s_3$ $n_2, n_4, n_5$
Sqr	$s_1, s_3$	$n_2, n_4$	$s_1, s_3$ $n_2, n_4$
Sub	$s_1, s_3, s_5$	$n_2, n_4, n_6$	$s_1, s_3, s_5, n_2, n_4, n_6$ $n_1$
Word_bytereverse		$n_1$	$n_1$
Word_clz		$n_1$	$n_1$
Word_ctz		$n_1$	$n_1$
Word_divstep59		$n_1$	$n_1, n_2, n_3, n_4$
Word_max		$n_1, n_2$	$n_1, n_2$
Word_min		$n_1, n_2$	$n_1, n_2$
Word_negmodinv		$n_1$	$n_1$
Word_recip		$n_1$	$n_1$

TOTAL VARIABLES: 93 179 85 187

# Quantitative Static Timing Analysis

Denis Mazzucato, Marco Campion,  
and Caterina Urban  
SAS 2024

Quantitative Static Timing Analysis

Denis Mazzucato<sup>[0000-0002-3613-2035]</sup>, Marco Campion<sup>[0000-0002-1099-3494]</sup>,  
and Caterina Urban<sup>[0000-0002-8127-9642]</sup>

INRIA & ENS | PSL,  
`{denis.mazzucato,marco.campion,caterina.urban}@inria.fr`

**Abstract.** Programming errors in software applications can often be difficult to detect, as they may appear without clear indications of failure. One such example is when certain input variables have an unexpected impact on the program's behavior. As an indicator of the program's runtime behavior, this work studies the impact of input variables on the number of loop iterations in a program. Such information is valuable for debugging, optimizing performance, and analyzing security vulnerabilities, such as in side-channel attacks where execution times can be exploited. To address this issue, we propose a sound static analysis based on abstract interpretation to quantify the impact of each input variable on the global number of iterations. Our approach combines a dependency analysis with a global loop bound analysis to derive an over-approximation of the impact quantity. We demonstrate our prototype tool in the S2N-BIGNUM library for cryptographic systems to certify the absence of timing side-channels.



**1 Introduction**

Detecting programming errors that yield a plausible yet incorrect or unsafe behavior is a challenging task. Particularly when these errors do not lead to obvious failures, like program crashes, but may degrade performance or introduce security vulnerabilities. One source of such errors may arise from the unexpected impact of input variables on the program's runtime. For instance, in the context of security, an unexpected impact of input variables on the program's runtime could reveal sensitive information [45], leading to potential security threats. Even in cryptography, where programs are mathematically robust, vulnerabilities to timing attacks persist, depending on the implementation choices and design. Kocher [21] demonstrated that widely used public key cryptographic algorithms, like RSA, are vulnerable to timing attacks, and may leak information about the secret key. The value of such attacks lies in their simplicity; attackers do not need



Radhia Cousot Award

# Intensional Properties

---

## Loop Iterations

# Conclusion

---

# Contributions

---

# Contributions

---

An Abstract Interpretation Framework for Input Data Usage

Caterina Urban and Peter Müller

ESOP 2018

Automation of Quantitative Information-Flow Analysis

Boris Köpf and Andrey Rybalchenko

SFM 2013

# Contributions

An Abstract Interpretation Framework for Input Data Usage  
Caterina Urban and Peter Müller  
ESOP 2018

Automation of Quantitative Information-Flow Analysis  
Boris Köpf and Andrey Rybalchenko  
SFM 2013

**Quantitative Input Usage Static Analysis**

Denis Mazzucato<sup>[0000-0002-3613-2035]</sup>, Marco Campion<sup>[0000-0002-1099-3494]</sup>, and Caterina Urban<sup>[0000-0002-8127-9642]</sup>  
INRIA & ENS | PSL,  
`{denis.mazzucato,marco.campion,caterina.urban}@inria.fr`

**Abstract.** Programming errors in software applications may produce plausible yet erroneous results, without providing a clear indication of failure. This opens, for instance, when an input variable has a disproportionate impact on the outcome. To address this issue, we propose a novel quantitative static analysis for determining the impact of inputs on the program computations, parametrized in the definition of impact. This static analysis employs an underlying abstract backward analyzer and computes a sound over-approximation of the impact of program inputs, providing valuable insights into how the analyzed program handles them. We implement a proof-of-concept static analyzer to demonstrate potential applications.

**1 Introduction**

Disastrous outcomes may result from programming errors in safety-critical settings, especially when they do not result in software failures but instead produce a plausible yet erroneous outcome. Such bugs are hard to spot since they provide no indication that something went wrong. A potential source of such errors is when an input variable has disproportionate impact on the program computations compared to the developers' expectations. A notable example is the Reinhart and Rogoff article "Growth in a Time of Debt" [19], which was heavily cited to justify austerity measures around the world in the following years, and was later discovered to be flawed [13]. Notably, one of the several programming and methodological errors discovered in the article is the incorrect usage of the input value relative to Norway's economic growth in 1964, compromising the authors' conclusion. Hence, it is important to employ techniques that enhance the confidence in the usage of input variables.

In this direction, Barowy et al. [2] proposed a stochastic approach specific for spreadsheet applications. Such approach is able to estimate the impact of input cells. However, the lack of mathematical guarantees precludes the employment of such technique in safety-critical contexts. On the other hand, existing formal methods-based approaches only target qualitative properties about input data usage, e.g., only addressing whether an input variable is used or not [22, 23].

In this work, we present a novel quantitative input usage framework to discriminate between input variables with different impact on the outcome of a

Quantitative Input Usage Static Analysis  
Denis Mazzucato, Marco Campion, and Caterina Urban  
NFM 2024

# Contributions

An Abstract Interpretation Framework for Input Data Usage  
Caterina Urban and Peter Müller  
ESOP 2018

Automation of Quantitative Information-Flow Analysis  
Boris Köpf and Andrey Rybalchenko  
SFM 2013

**Quantitative Input Usage Static Analysis**

Denis Mazzucato<sup>[0000-0002-3613-2035]</sup>, Marco Campion<sup>[0000-0002-1099-3494]</sup>, and Caterina Urban<sup>[0000-0002-8127-9642]</sup>  
INRIA & ENS | PSL,  
`{denis.mazzucato,marco.campion,caterina.urban}@inria.fr`

**Abstract.** Programming errors in software applications may produce plausible yet erroneous results, without providing a clear indication of failure. This opens, for instance, when an input variable has a disproportionate impact on the program's outcome. To address this issue, we propose a novel quantitative static analysis for determining the impact of inputs on the program computations, parametrized in the definition of impact. This static analysis employs an underlying abstract backward analyzer and computes a sound over-approximation of the impact of program inputs, providing valuable insights into how the analyzed program handles them. We implement a proof-of-concept static analyzer to demonstrate potential applications.

**1 Introduction**

Disastrous outcomes may result from programming errors in safety-critical settings, especially when they do not result in software failures but instead produce a plausible yet erroneous outcome. Such bugs are hard to spot since they provide no indication that something went wrong. A potential source of such errors is when an input variable has disproportionate impact on the program computations compared to the developers' expectations. A notable example is the Reinhart and Rogoff article "Growth in a Time of Debt" [19], which was heavily cited to justify austerity measures around the world in the following years, and was later discovered to be flawed [13]. Notably, one of the several programming and methodological errors discovered in the article is the incorrect usage of the input value relative to Norway's economic growth in 1964, compromising the authors' conclusion. Hence, it is important to employ techniques that enhance the confidence in the usage of input variables.

In this direction, Barowy et al. [2] proposed a stochastic approach specific for spreadsheet applications. Such approach is able to estimate the impact of input cells. However, the lack of mathematical guarantees precludes the employment of such technique in safety-critical contexts. On the other hand, existing formal methods-based approaches only target qualitative properties about input data usage, e.g., only addressing whether an input variable is used or not [22, 23].

In this work, we present a novel quantitative input usage framework to discriminate between input variables with different impact on the outcome of a

Quantitative Input Usage Static Analysis  
Denis Mazzucato, Marco Campion, and Caterina Urban  
NFM 2024

- Quantitative Framework

# Contributions

An Abstract Interpretation Framework for Input Data Usage  
Caterina Urban and Peter Müller  
ESOP 2018

Automation of Quantitative Information-Flow Analysis  
Boris Köpf and Andrey Rybalchenko  
SFM 2013

**Quantitative Input Usage Static Analysis**

Denis Mazzucato<sup>[0000-0002-3613-2035]</sup>, Marco Campion<sup>[0000-0002-1099-3494]</sup>, and Caterina Urban<sup>[0000-0002-8127-9642]</sup>  
INRIA & ENS | PSL,  
`{denis.mazzucato,marco.campion,caterina.urban}@inria.fr`

**Abstract.** Programming errors in software applications may produce plausible yet erroneous results, without providing a clear indication of failure. This opens, for instance, when an input variable has a disproportionate impact on the program's outcome. To address this issue, we propose a novel quantitative static analysis for determining the impact of inputs on the program computations, parametrized in the definition of impact. This static analysis employs an underlying abstract backward analyzer and computes a sound over-approximation of the impact of program inputs, providing valuable insights into how the analyzed program handles them. We implement a proof-of-concept static analyzer to demonstrate potential applications.

**1 Introduction**

Disastrous outcomes may result from programming errors in safety-critical settings, especially when they do not result in software failures but instead produce a plausible yet erroneous outcome. Such bugs are hard to spot since they provide no indication that something went wrong. A potential source of such errors is when an input variable has disproportionate impact on the program computations compared to the developers' expectations. A notable example is the Reinhart and Rogoff article "Growth in a Time of Debt" [19], which was heavily cited to justify austerity measures around the world in the following years, and was later discovered to be flawed [13]. Notably, one of the several programming and methodological errors discovered in the article is the incorrect usage of the input value relative to Norway's economic growth in 1964, compromising the authors' conclusion. Hence, it is important to employ techniques that enhance the confidence in the usage of input variables.

In this direction, Barowy et al. [2] proposed a stochastic approach specific for spreadsheet applications. Such approach is able to estimate the impact of input cells. However, the lack of mathematical guarantees precludes the employment of such technique in safety-critical contexts. On the other hand, existing formal methods-based approaches only target qualitative properties about input data usage, e.g., only addressing whether an input variable is used or not [22, 23].

In this work, we present a novel quantitative input usage framework to discriminate between input variables with different impact on the outcome of a

Quantitative Input Usage Static Analysis  
Denis Mazzucato, Marco Campion, and Caterina Urban  
NFM 2024

- Quantitative Framework
- 3 different Quantifiers

# Contributions

An Abstract Interpretation Framework for Input Data Usage  
Caterina Urban and Peter Müller  
ESOP 2018

Automation of Quantitative Information-Flow Analysis  
Boris Köpf and Andrey Rybalchenko  
SFM 2013

**Quantitative Input Usage Static Analysis**

Denis Mazzucato<sup>[0000-0002-3613-2035]</sup>, Marco Campion<sup>[0000-0002-1099-3494]</sup>, and Caterina Urban<sup>[0000-0002-8127-9642]</sup>  
INRIA & ENS | PSL,  
`{denis.mazzucato,marco.campion,caterina.urban}@inria.fr`

**Abstract.** Programming errors in software applications may produce plausible yet erroneous results, without providing a clear indication of failure. This opens, for instance, when an input variable has a disproportionate impact on the outcome. To address this issue, we propose a novel quantitative static analysis for determining the impact of inputs on the program computations, parametrized in the definition of impact. This static analysis employs an underlying abstract backward analyzer and computes a sound over-approximation of the impact of program inputs, providing valuable insights into how the analyzed program handles them. We implement a proof-of-concept static analyzer to demonstrate potential applications.

**1 Introduction**

Disastrous outcomes may result from programming errors in safety-critical settings, especially when they do not result in software failures but instead produce a plausible yet erroneous outcome. Such bugs are hard to spot since they provide no indication that something went wrong. A potential source of such errors is when an input variable has disproportionate impact on the program computations compared to the developers' expectations. A notable example is the Reinhart and Rogoff article "Growth in a Time of Debt" [19], which was heavily cited to justify austerity measures around the world in the following years, and was later discovered to be flawed [13]. Notably, one of the several programming and methodological errors discovered in the article is the incorrect usage of the input value relative to Norway's economic growth in 1964, compromising the authors' conclusion. Hence, it is important to employ techniques that enhance the confidence in the usage of input variables.

In this direction, Barowy et al. [2] proposed a stochastic approach specific for spreadsheet applications. Such approach is able to estimate the impact of input cells. However, the lack of mathematical guarantees precludes the employment of such technique in safety-critical contexts. On the other hand, existing formal methods-based approaches only target qualitative properties about input data usage, e.g., only addressing whether an input variable is used or not [22, 23].

In this work, we present a novel quantitative input usage framework to discriminate between input variables with different impact on the outcome of a

Quantitative Input Usage Static Analysis  
Denis Mazzucato, Marco Campion, and Caterina Urban  
NFM 2024

- Quantitative Framework
- 3 different Quantifiers
- IMPATTO

# Contributions

An Abstract Interpretation Framework for Input Data Usage  
Caterina Urban and Peter Müller  
ESOP 2018

Automation of Quantitative Information-Flow Analysis  
Boris Köpf and Andrey Rybalchenko  
SFM 2013

**Quantitative Input Usage Static Analysis**

Denis Mazzucato<sup>[0000-0002-3613-2035]</sup>, Marco Campion<sup>[0000-0002-1099-3494]</sup>, and Caterina Urban<sup>[0000-0002-8127-9642]</sup>  
INRIA & ENS | PSL,  
`{denis.mazzucato,marco.campion,caterina.urban}@inria.fr`

**Abstract.** Programming errors in software applications may produce plausible yet erroneous results, without providing a clear indication of failure. This opens, for instance, when a variable has a disproportionate impact on the program. To address this issue, we propose a novel quantitative static analysis for determining the impact of inputs on the program computations, parametrized in the definition of impact. This static analysis employs an underlying abstract backward analyzer and computes a sound over-approximation of the impact of program inputs, providing valuable insights into how the analyzed program handles them. We implement a proof-of-concept static analyzer to demonstrate potential applications.

**1 Introduction**

Disastrous outcomes may result from programming errors in safety-critical settings, especially when they do not result in software failures but instead produce a plausible yet erroneous outcome. Such bugs are hard to spot since they provide no indication that something went wrong. A potential source of such errors is when an input variable has disproportionate impact on the program computations compared to the developers' expectations. A notable example is the Reinhart and Rogoff article "Growth in a Time of Debt" [19], which was heavily cited to justify austerity measures around the world in the following years, and was later discovered to be flawed [13]. Notably, one of the several programming and methodological errors discovered in the article is the incorrect usage of the input value relative to Norway's economic growth in 1964, compromising the authors' conclusion. Hence, it is important to employ techniques that enhance the confidence in the usage of input variables.

In this direction, Barowy et al. [2] proposed a stochastic approach specific for spreadsheet applications. Such approach is able to estimate the impact of input cells. However, the lack of mathematical guarantees precludes the employment of such technique in safety-critical contexts. On the other hand, existing formal methods-based approaches only target qualitative properties about input data usage, e.g., only addressing whether an input variable is used or not [22, 23].

In this work, we present a novel quantitative input usage framework to discriminate between input variables with different impact on the outcome of a

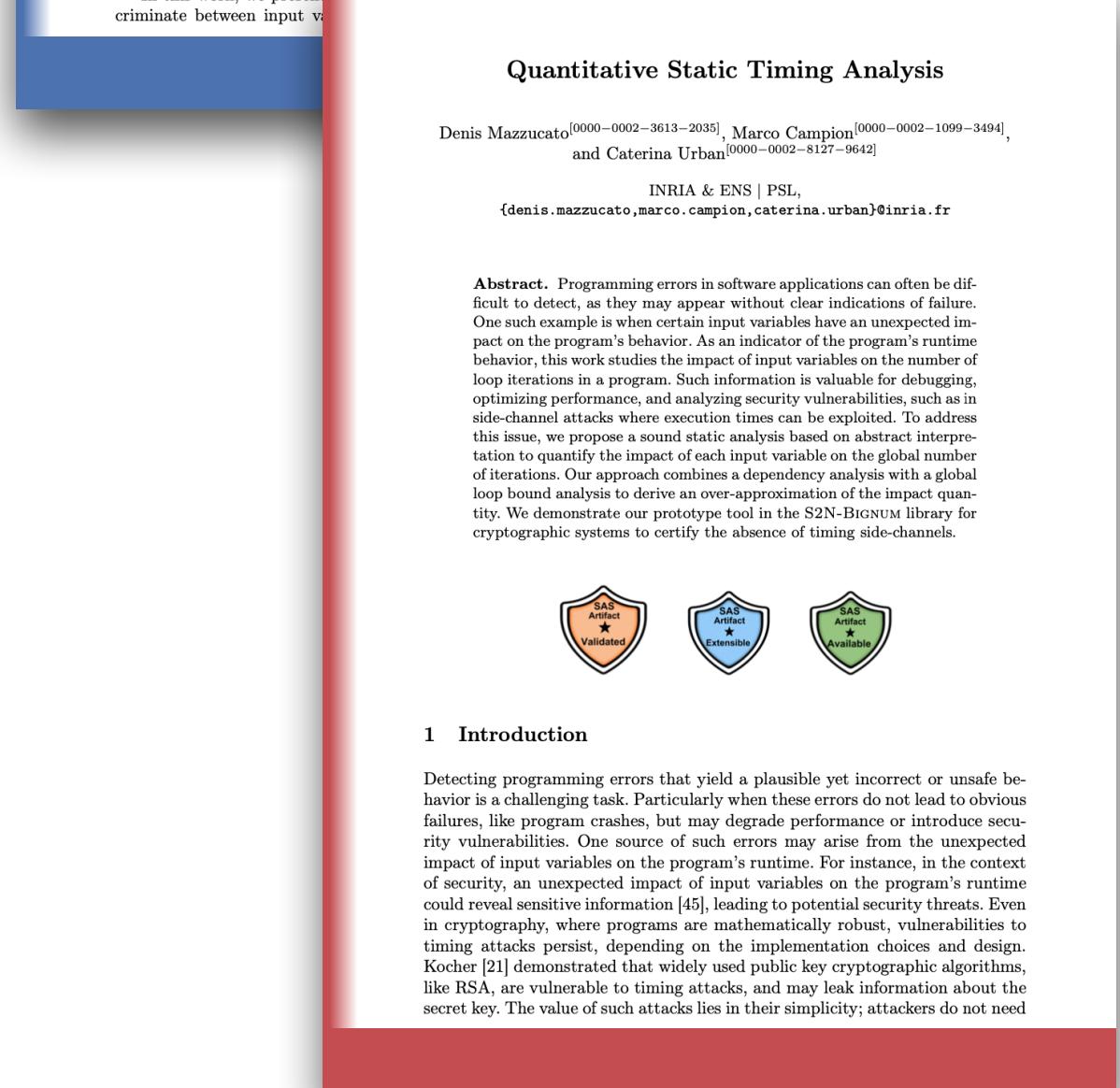
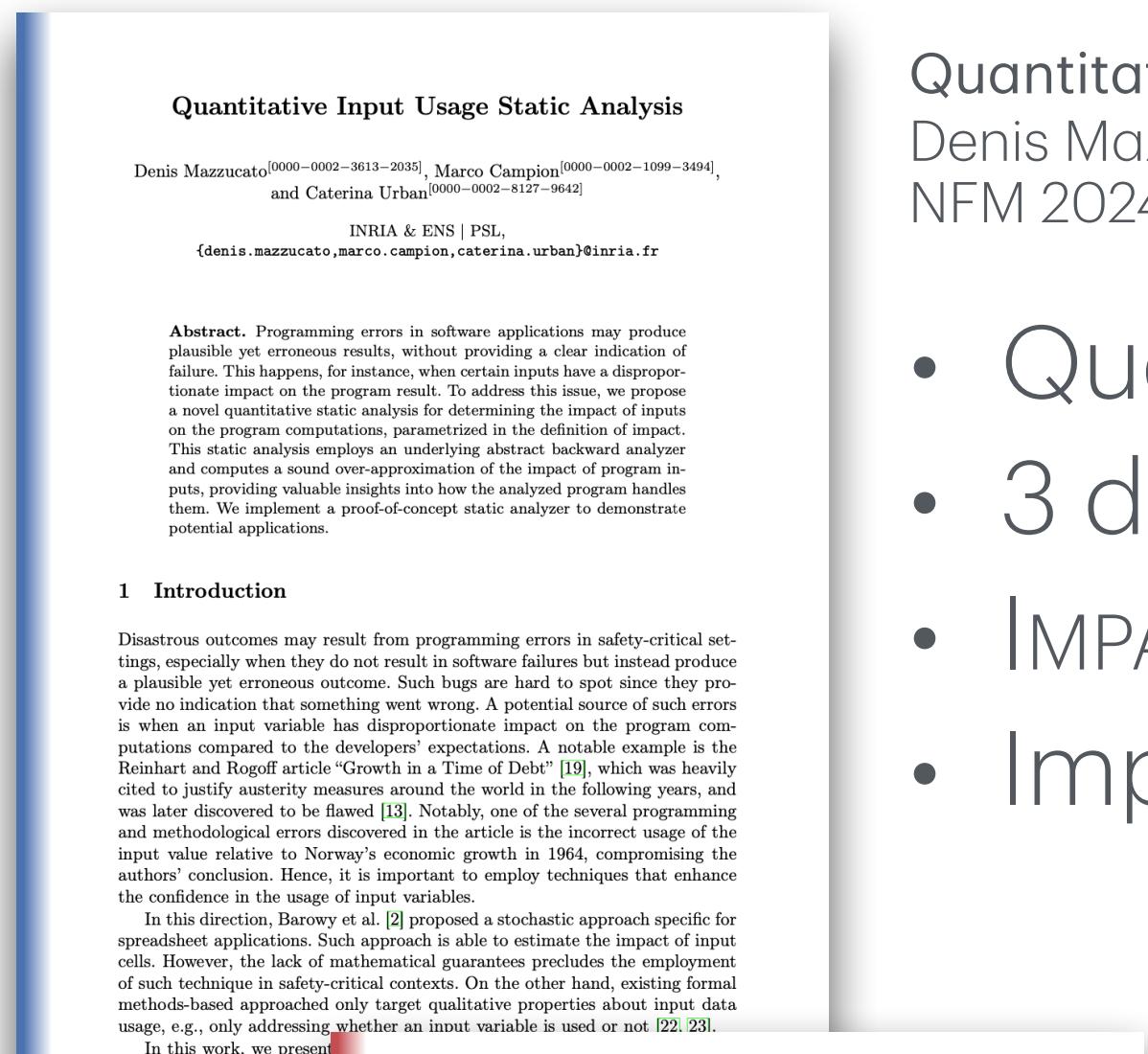
Quantitative Input Usage Static Analysis  
Denis Mazzucato, Marco Campion, and Caterina Urban  
NFM 2024

- Quantitative Framework
- 3 different Quantifiers
- IMPATTO
- Impact of a few use cases

# Contributions

An Abstract Interpretation Framework for Input Data Usage  
Caterina Urban and Peter Müller  
ESOP 2018

Automation of Quantitative Information-Flow Analysis  
Boris Köpf and Andrey Rybalchenko  
SFM 2013



Quantitative Input Usage Static Analysis  
Denis Mazzucato, Marco Campion, and Caterina Urban  
NFM 2024

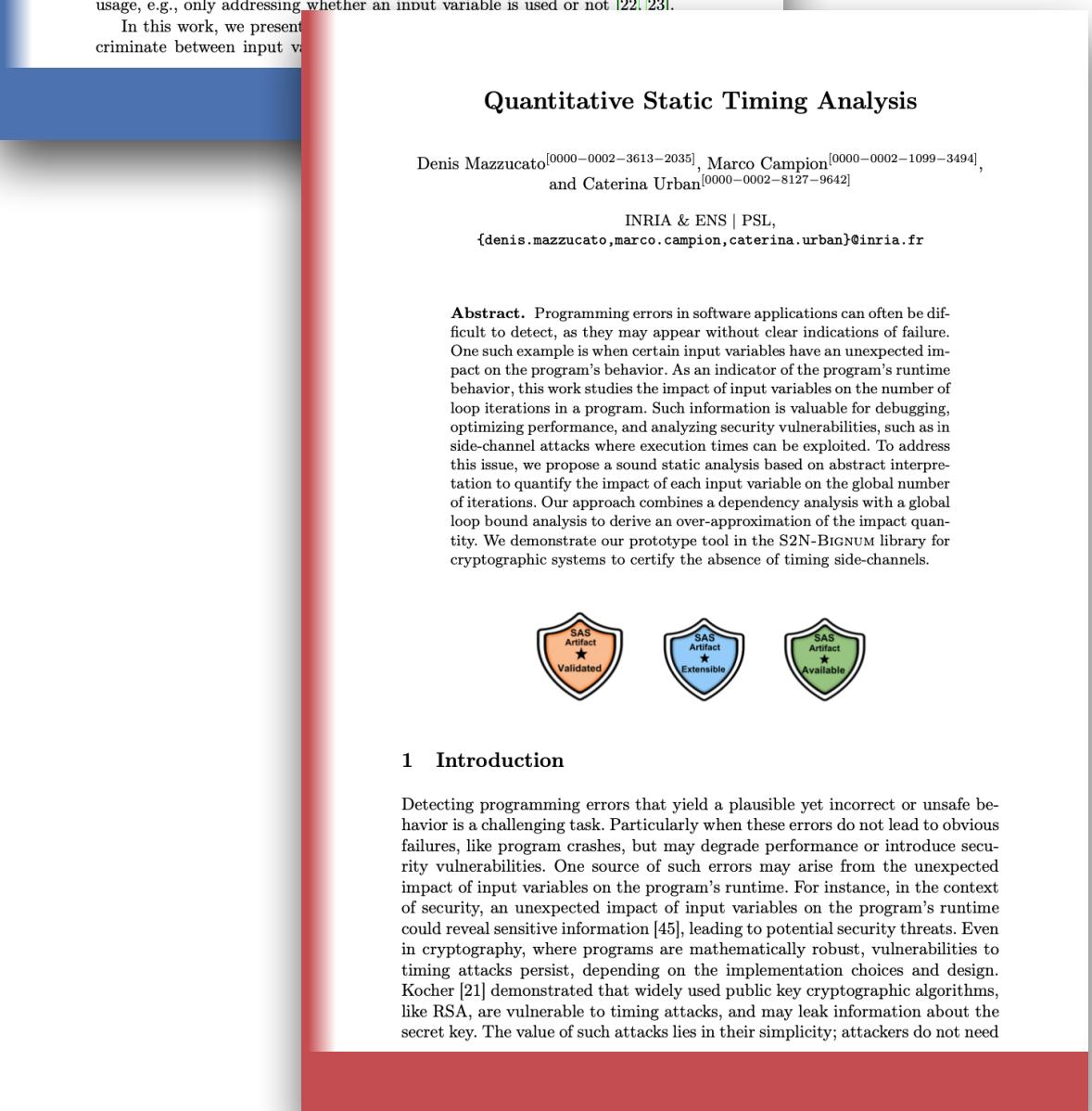
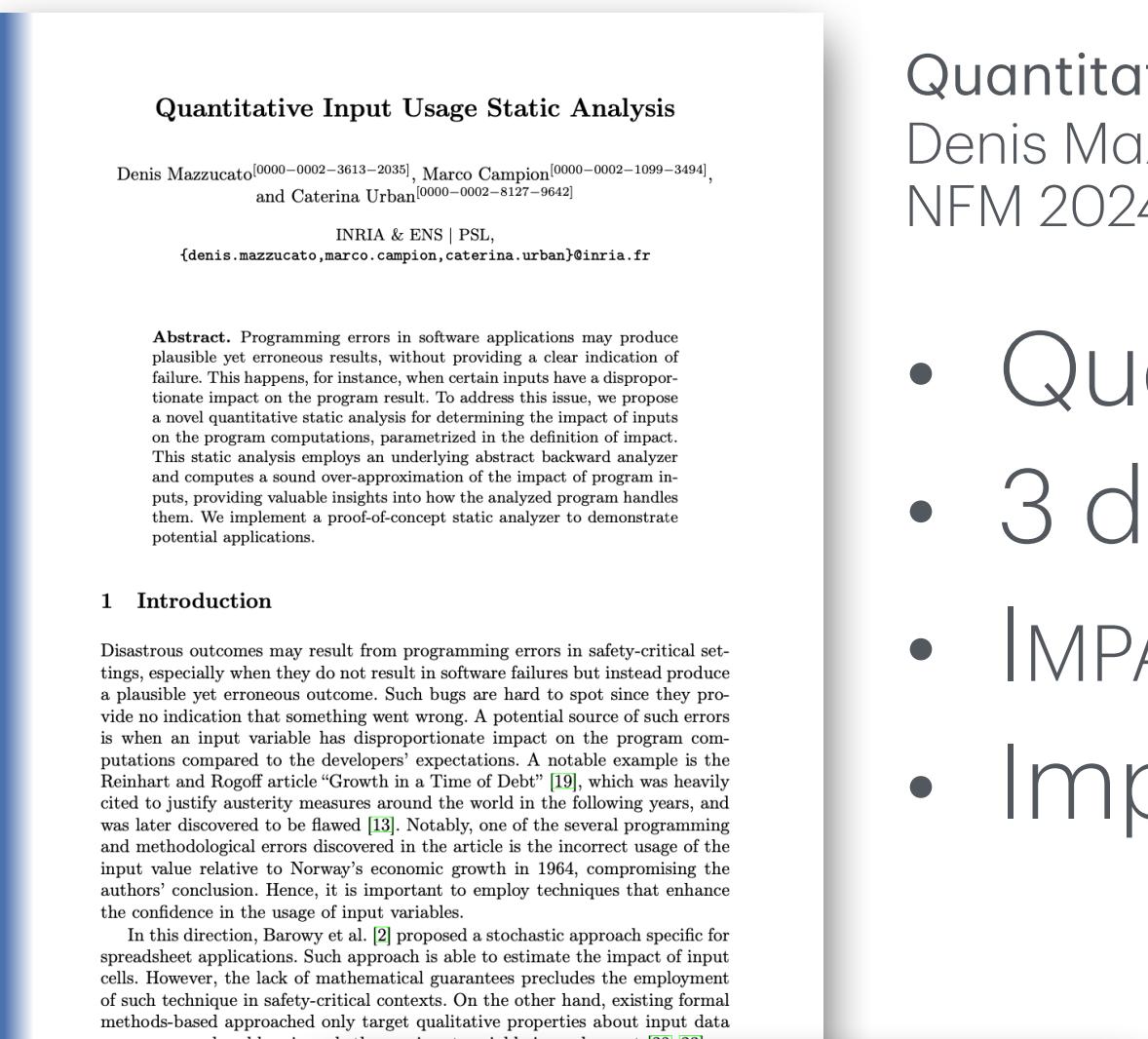
- Quantitative Framework
- 3 different Quantifiers
- IMPATTO
- Impact of a few use cases

Quantitative Static Timing Analysis  
Denis Mazzucato, Marco Campion,  
and Caterina Urban  
SAS 2024

# Contributions

An Abstract Interpretation Framework for Input Data Usage  
Caterina Urban and Peter Müller  
ESOP 2018

Automation of Quantitative Information-Flow Analysis  
Boris Köpf and Andrey Rybalchenko  
SFM 2013



Quantitative Input Usage Static Analysis  
Denis Mazzucato, Marco Campion, and Caterina Urban  
NFM 2024

- Quantitative Framework
- 3 different Quantifiers
- IMPATTO
- Impact of a few use cases

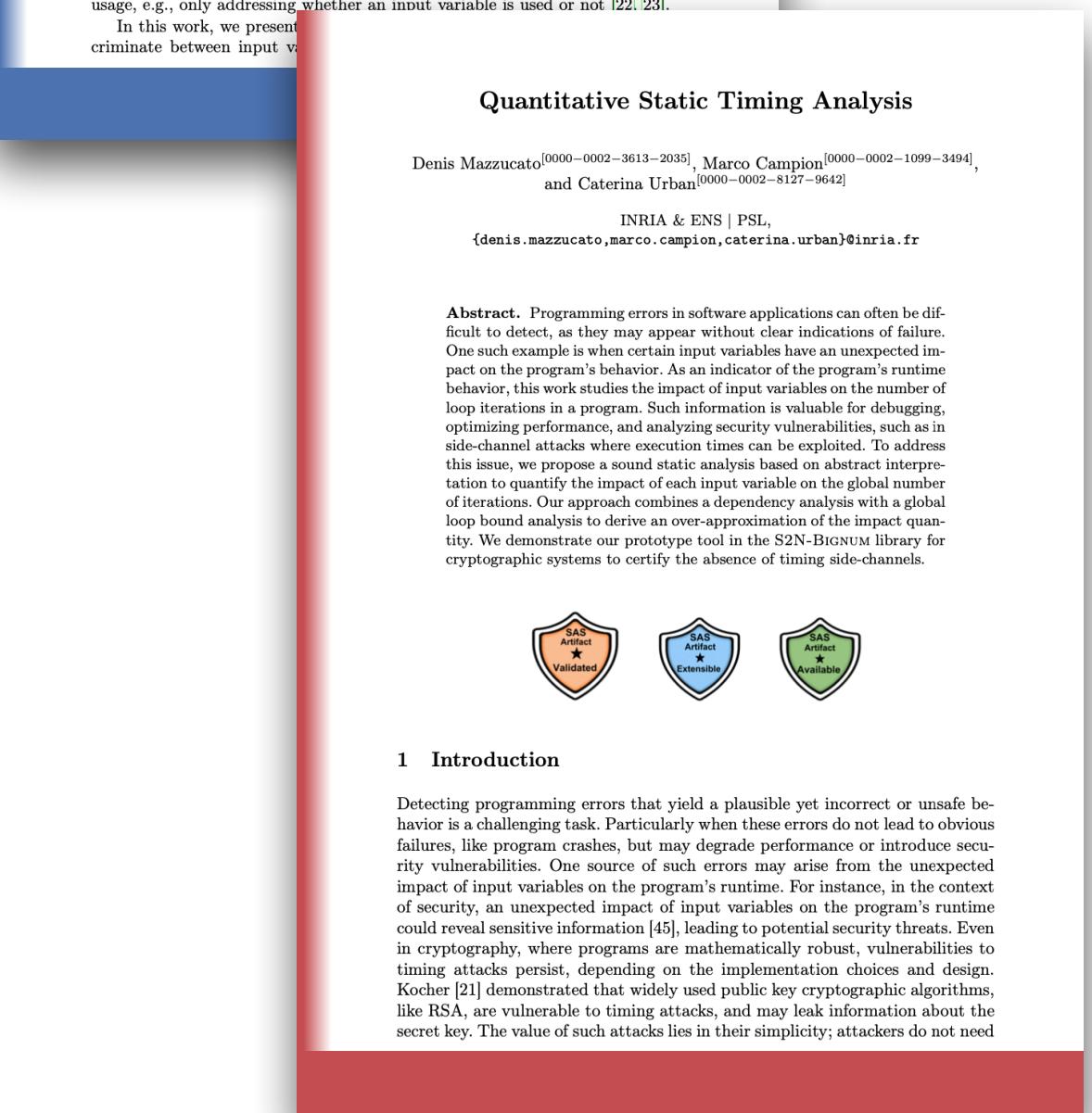
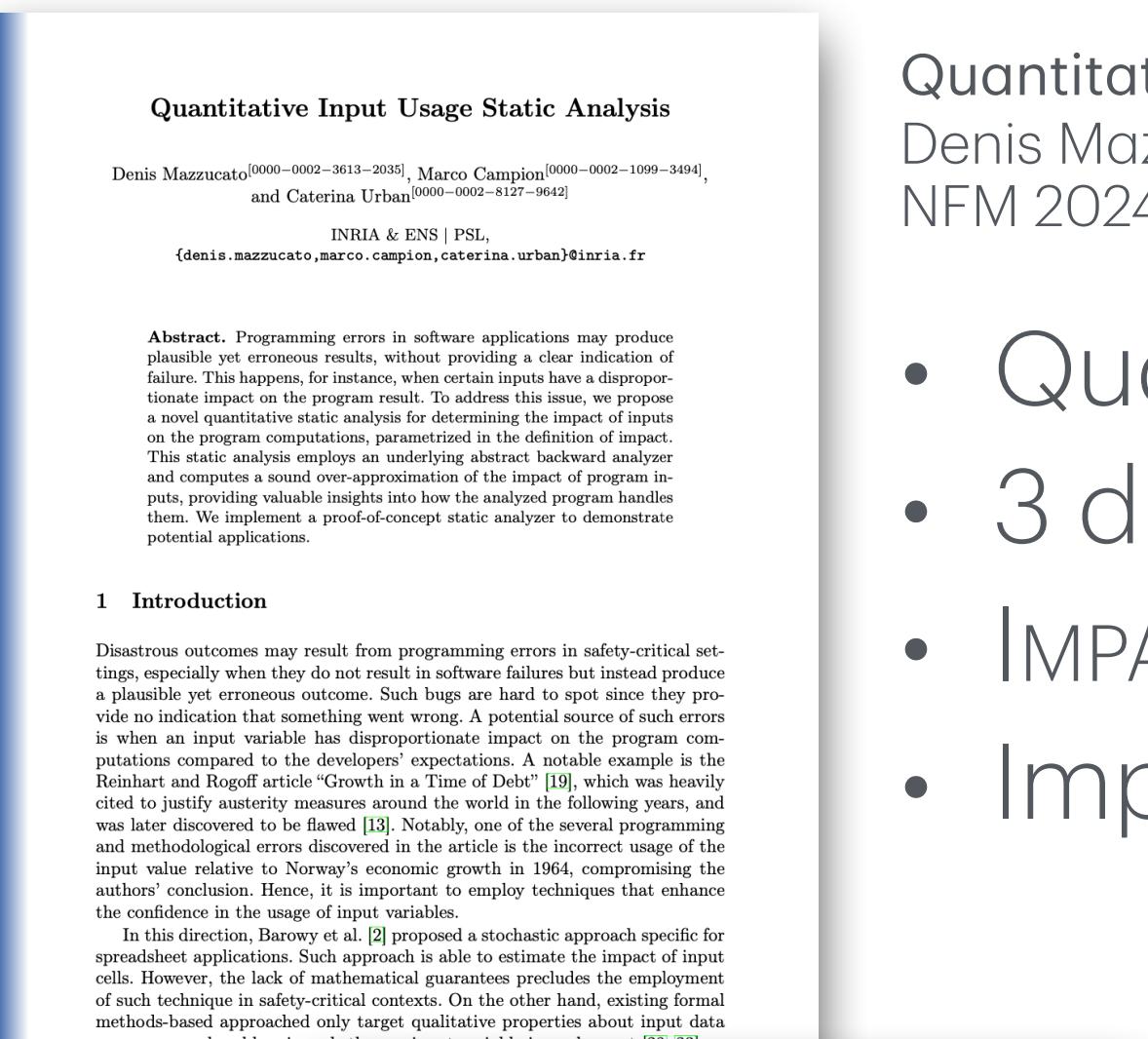
Quantitative Static Timing Analysis  
Denis Mazzucato, Marco Campion,  
and Caterina Urban  
SAS 2024

- Intensional Property

# Contributions

An Abstract Interpretation Framework for Input Data Usage  
Caterina Urban and Peter Müller  
ESOP 2018

Automation of Quantitative Information-Flow Analysis  
Boris Köpf and Andrey Rybalchenko  
SFM 2013



Quantitative Input Usage Static Analysis  
Denis Mazzucato, Marco Campion, and Caterina Urban  
NFM 2024

- Quantitative Framework
- 3 different Quantifiers
- IMPATTO
- Impact of a few use cases

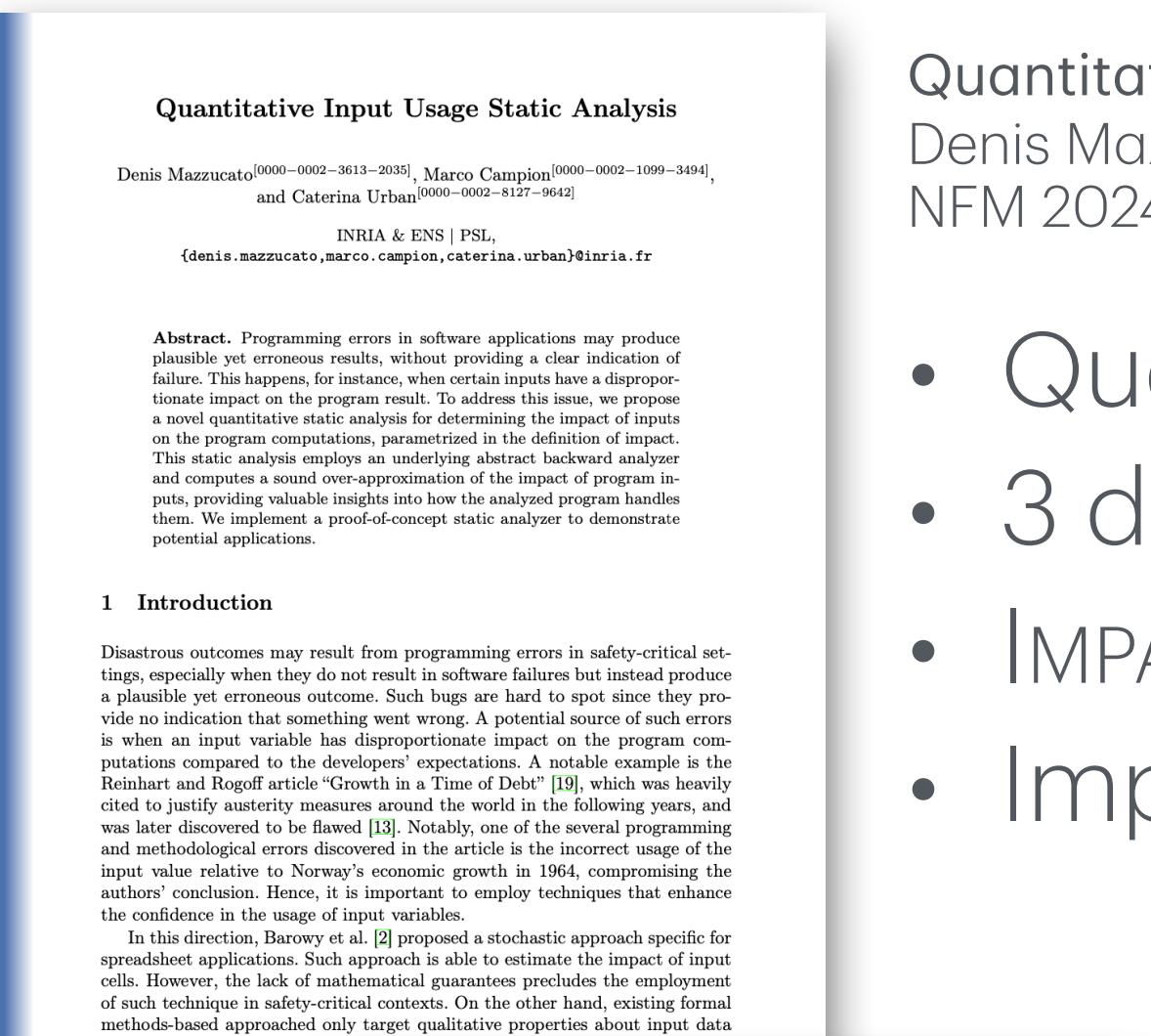
Quantitative Static Timing Analysis  
Denis Mazzucato, Marco Campion,  
and Caterina Urban  
SAS 2024

- Intensional Property
- Optimized tool TIMESEC

# Contributions

An Abstract Interpretation Framework for Input Data Usage  
Caterina Urban and Peter Müller  
ESOP 2018

Automation of Quantitative Information-Flow Analysis  
Boris Köpf and Andrey Rybalchenko  
SFM 2013



Quantitative Input Usage Static Analysis  
Denis Mazzucato, Marco Campion, and Caterina Urban  
NFM 2024

- Quantitative Framework
- 3 different Quantifiers
- IMPATTO
- Impact of a few use cases

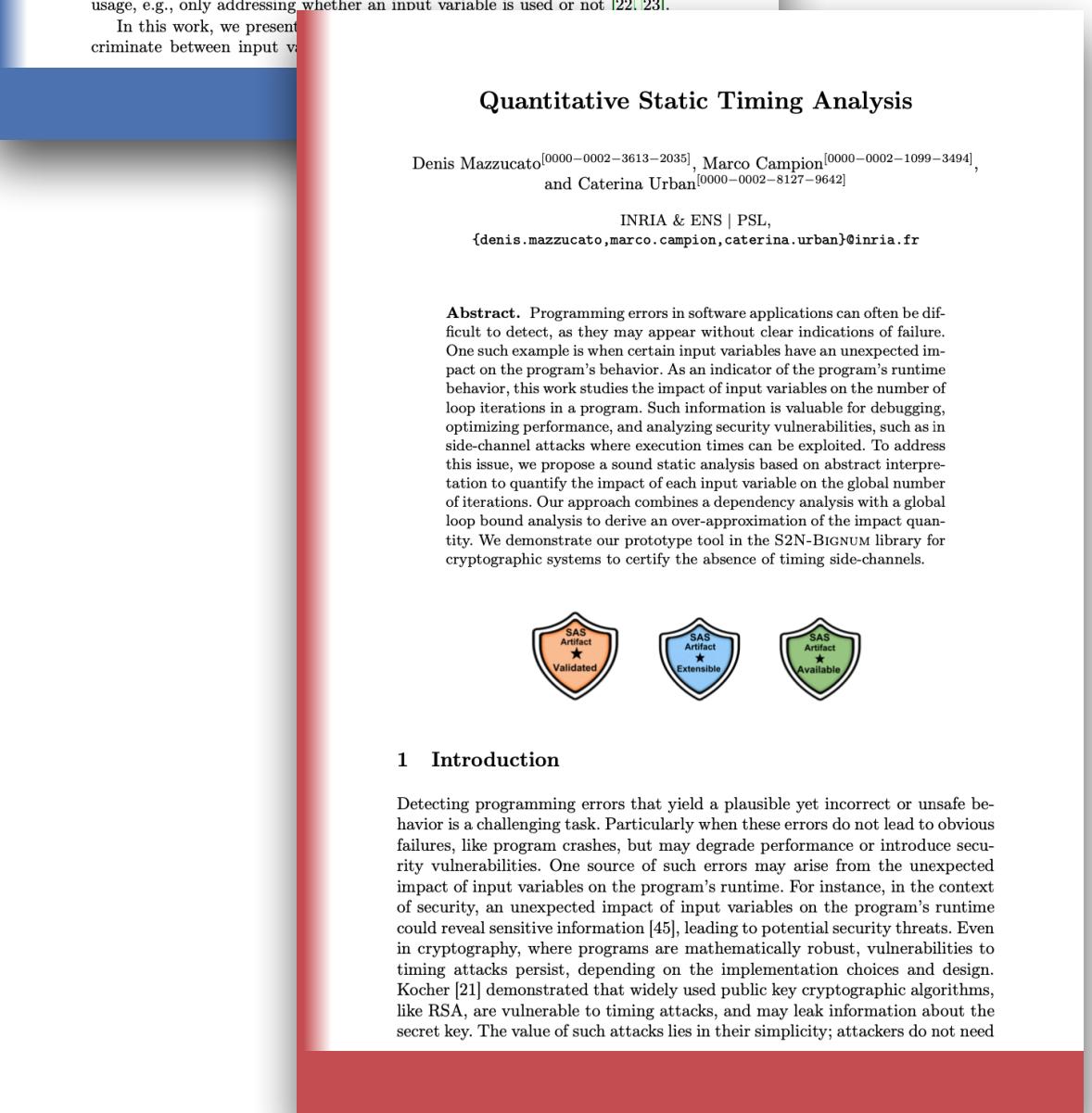
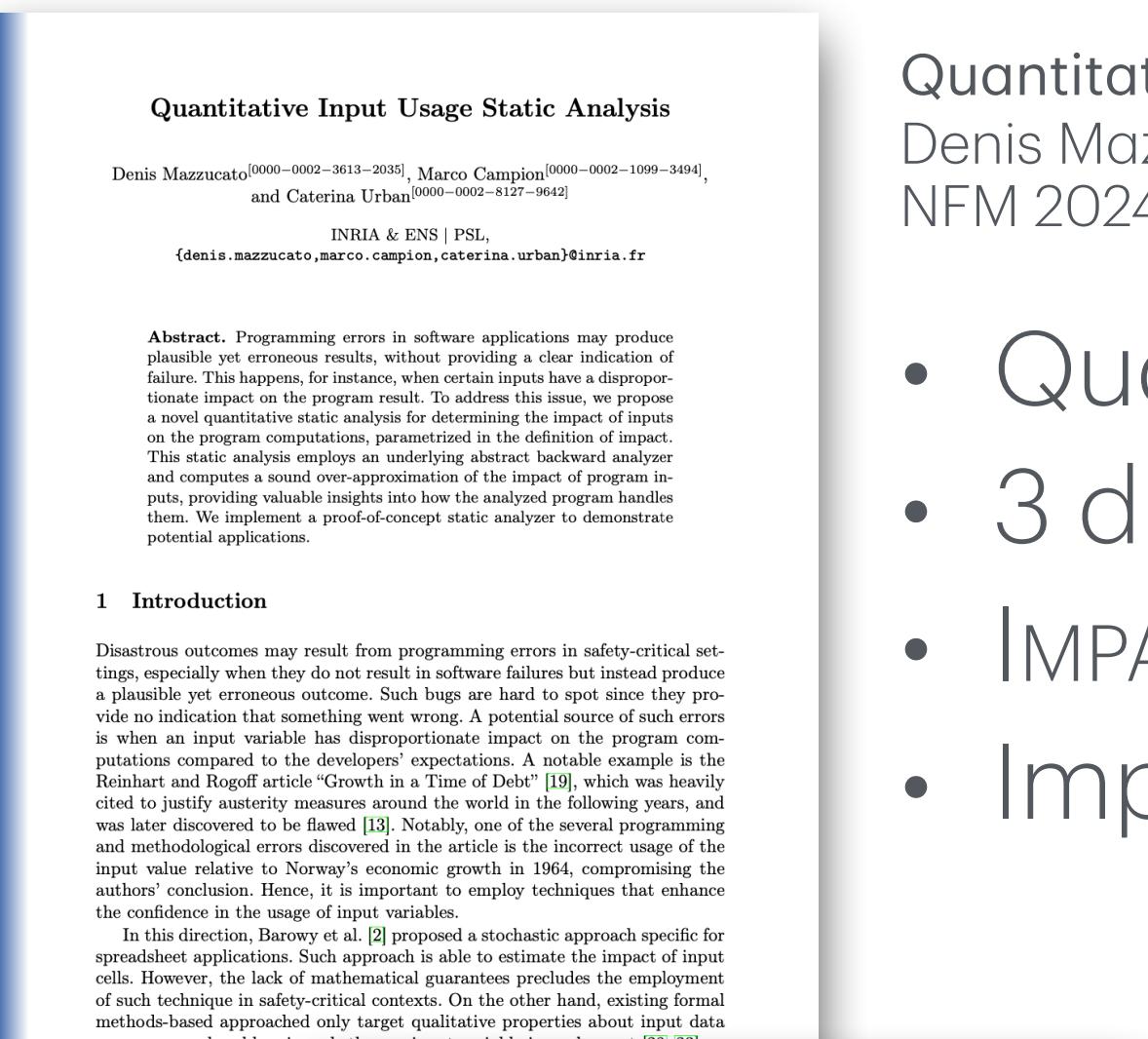
Quantitative Static Timing Analysis  
Denis Mazzucato, Marco Campion,  
and Caterina Urban  
SAS 2024

- Intensional Property
- Optimized tool TIMESEC
- AWS s2n-bignum

# Contributions

An Abstract Interpretation Framework for Input Data Usage  
Caterina Urban and Peter Müller  
ESOP 2018

Automation of Quantitative Information-Flow Analysis  
Boris Köpf and Andrey Rybalchenko  
SFM 2013



Quantitative Input Usage Static Analysis  
Denis Mazzucato, Marco Campion, and Caterina Urban  
NFM 2024

- Quantitative Framework
- 3 different Quantifiers
- IMPATTO
- Impact of a few use cases

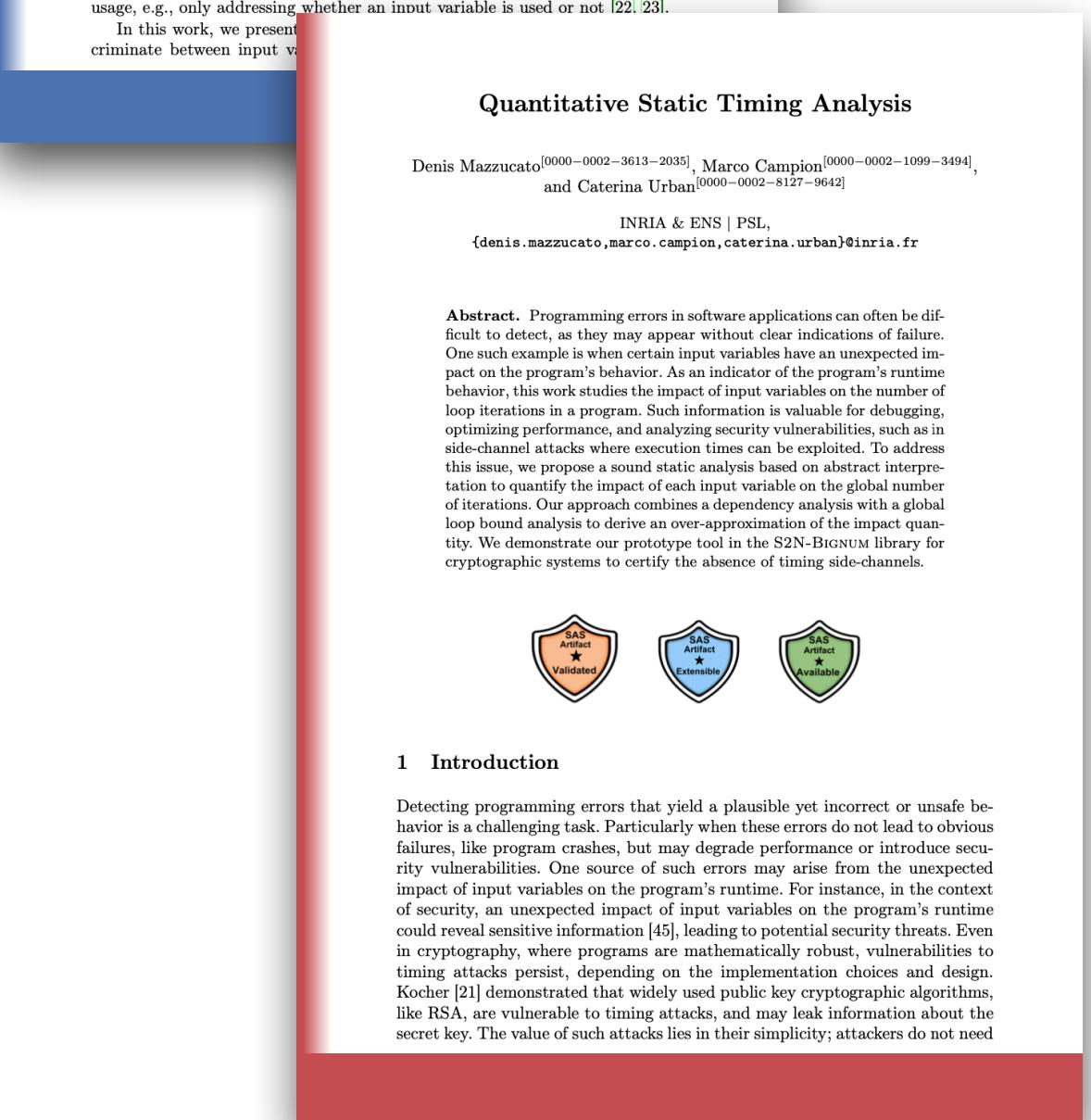
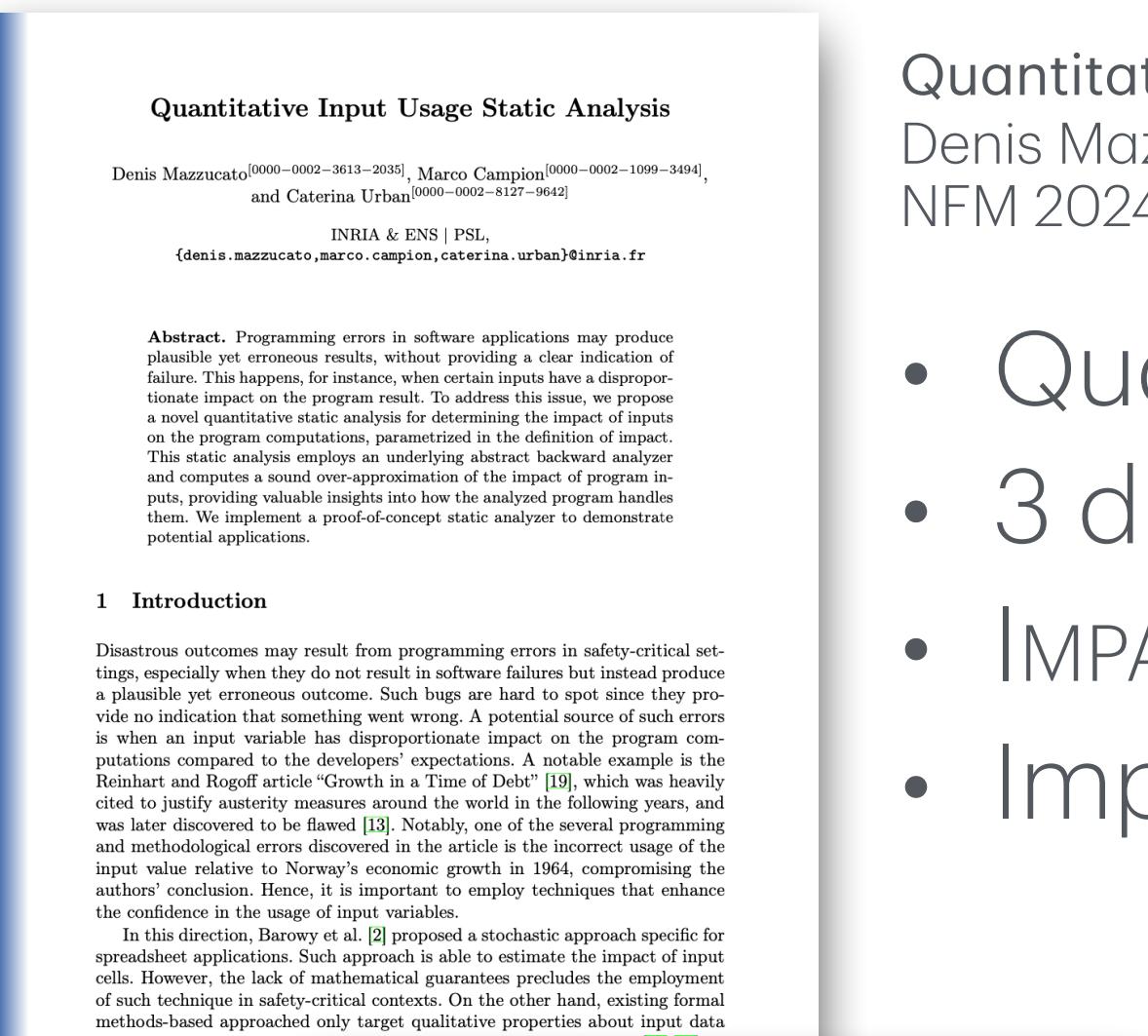
Quantitative Static Timing Analysis  
Denis Mazzucato, Marco Campion,  
and Caterina Urban  
SAS 2024

- Intensional Property
- Optimized tool TIMESEC
- AWS s2n-bignum
- SVComp

# Contributions

An Abstract Interpretation Framework for Input Data Usage  
Caterina Urban and Peter Müller  
ESOP 2018

Automation of Quantitative Information-Flow Analysis  
Boris Köpf and Andrey Rybalchenko  
SFM 2013



Quantitative Input Usage Static Analysis  
Denis Mazzucato, Marco Campion, and Caterina Urban  
NFM 2024

- Quantitative Framework
- 3 different Quantifiers
- IMPATTO
- Impact of a few use cases

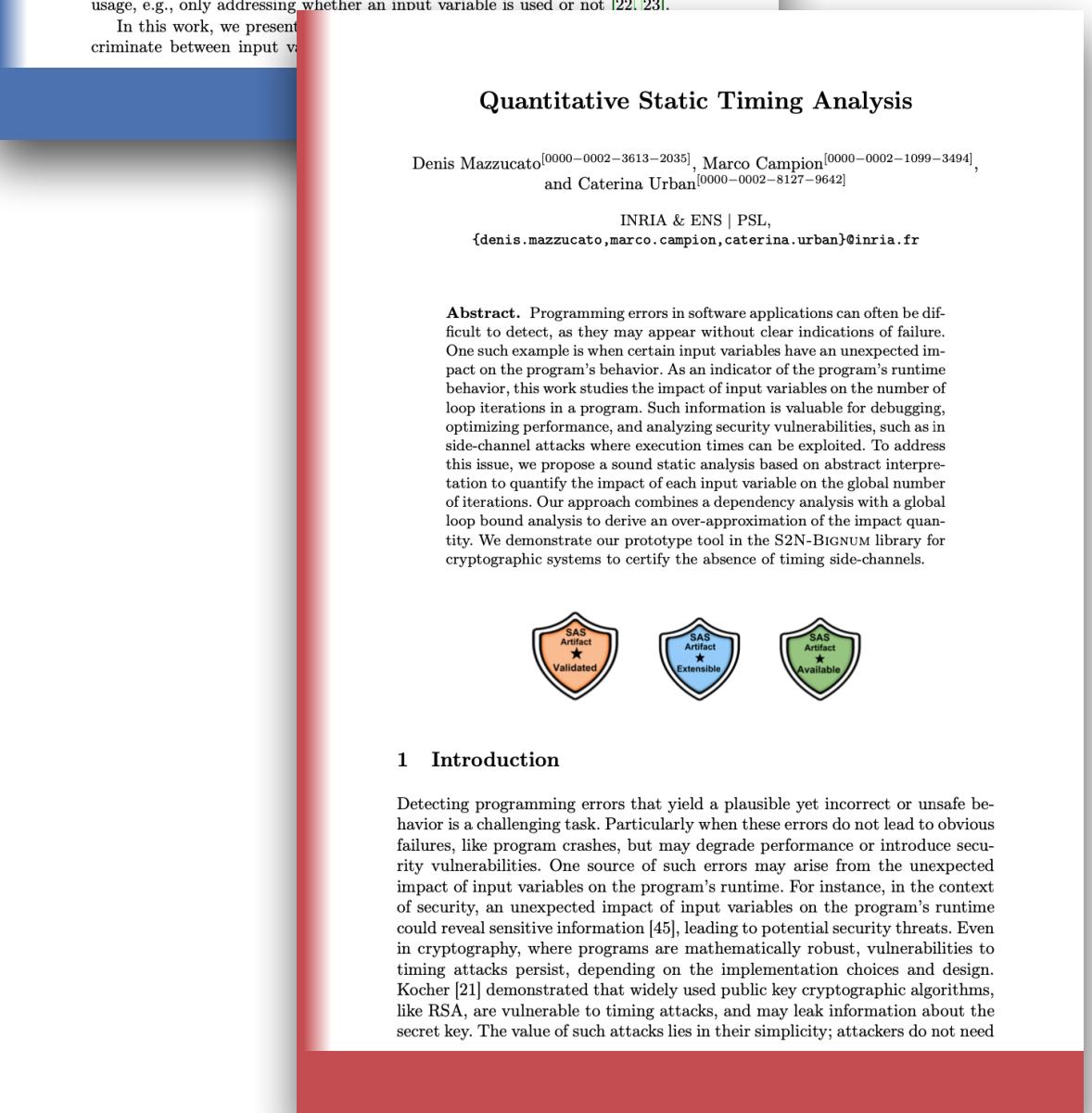
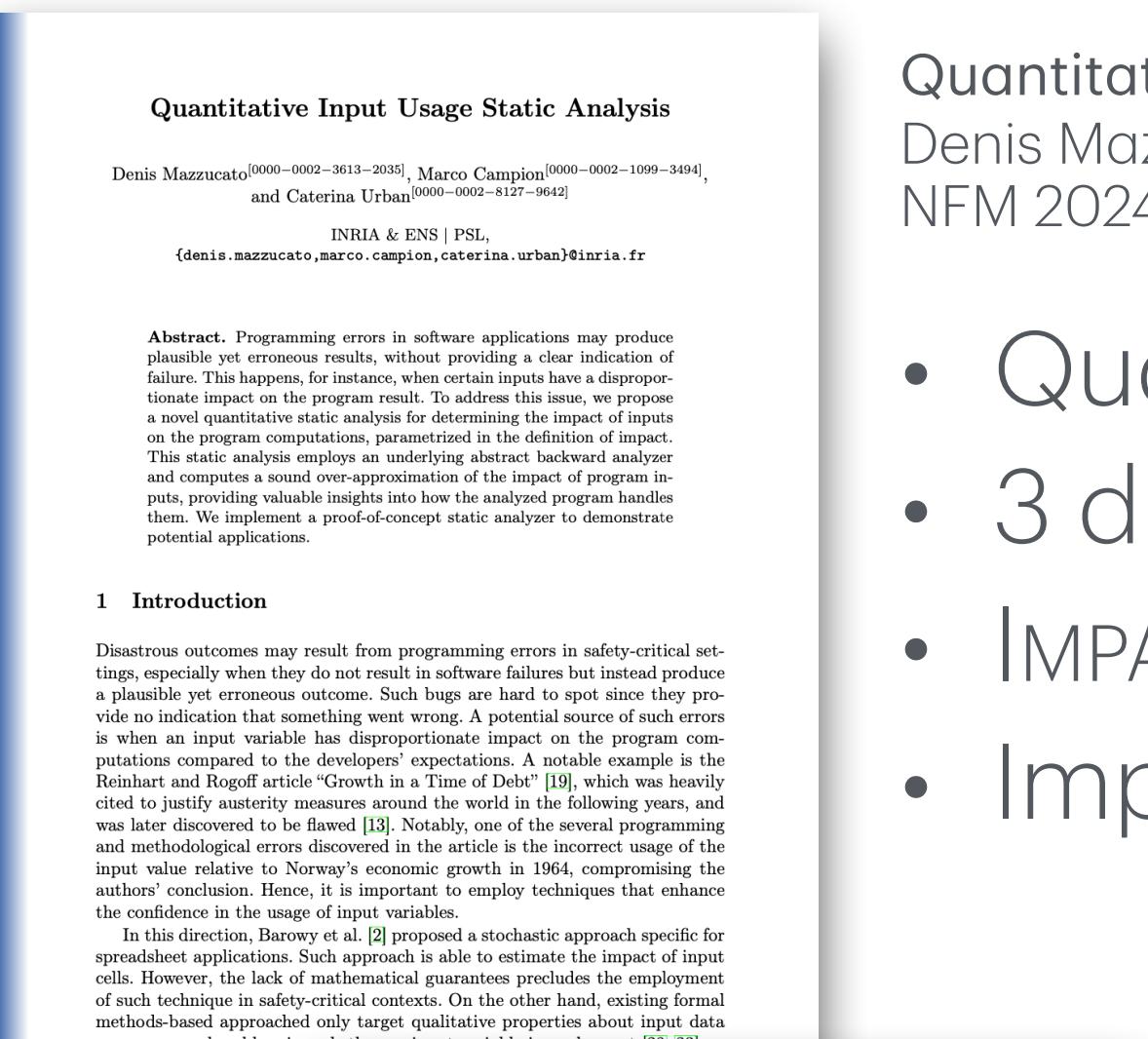
Quantitative Static Timing Analysis  
Denis Mazzucato, Marco Campion,  
and Caterina Urban  
SAS 2024

- Intensional Property
- Optimized tool TIMESEC
- AWS s2n-bignum
- SVComp
- Artifact

# Contributions

An Abstract Interpretation Framework for Input Data Usage  
Caterina Urban and Peter Müller  
ESOP 2018

Automation of Quantitative Information-Flow Analysis  
Boris Köpf and Andrey Rybalchenko  
SFM 2013



Quantitative Input Usage Static Analysis  
Denis Mazzucato, Marco Campion, and Caterina Urban  
NFM 2024

- Quantitative Framework
- 3 different Quantifiers
- IMPATTO
- Impact of a few use cases

Quantitative Static Timing Analysis  
Denis Mazzucato, Marco Campion,  
and Caterina Urban  
SAS 2024

- Intensional Property
- Optimized tool TIMESEC
- AWS s2n-bignum
- SVComp
- Artifact

# Contributions

An Abstract Interpretation Framework for Input Data Usage  
Caterina Urban and Peter Müller  
ESOP 2018

Automation of Quantitative Information-Flow Analysis  
Boris Köpf and Andrey Rybalchenko  
SFM 2013

Perfectly Parallel Fairness Certification of Neural Networks  
Caterina Urban et al.  
OOPSLA 2020

**Quantitative Input Usage Static Analysis**  
Denis Mazzucato<sup>[0000-0002-3613-2035]</sup>, Marco Campion<sup>[0000-0002-1099-3494]</sup>, and Caterina Urban<sup>[0000-0002-8127-9642]</sup>  
INRIA & ENS | PSL,  
`{denis.mazzucato,marco.campion,caterina.urban}@inria.fr`

**Abstract.** Programming errors in software applications may produce plausible yet erroneous results, without providing a clear indication of failure. This happens, for instance, when an input variable has a disproportionate impact on the program's behavior. To address this issue, we propose a novel quantitative static analysis for determining the impact of inputs on the program computations, parameterized in the definition of impact. This static analysis employs an underlying abstract backward analyzer and computes a sound over-approximation of the impact of program inputs, providing valuable insights into how the analyzed program handles them. We implement a proof-of-concept static analyzer to demonstrate potential applications.

**1 Introduction**  
Disastrous outcomes may result from programming errors in safety-critical settings, especially when they do not result in software failures but instead produce a plausible yet erroneous outcome. Such bugs are hard to spot since they provide no indication that something went wrong. A potential source of such errors is when an input variable has disproportionate impact on the program computations compared to the developers' expectations. A notable example is the Reinhart and Rogoff article "Growth in a Time of Debt" [19], which was heavily cited to justify austerity measures around the world in the following years, and was later discovered to be flawed [13]. Notably, one of the several programming and methodological errors discovered in the article is the incorrect usage of the input value relative to Norway's economic growth in 1964, compromising the authors' conclusion. Hence, it is important to employ techniques that enhance the confidence in the usage of input variables.

In this direction, Barowy et al. [2] proposed a stochastic approach specific for spreadsheet applications. Such approach is able to estimate the impact of input cells. However, the lack of mathematical guarantees precludes the employment of such technique in safety-critical contexts. On the other hand, existing formal methods-based approaches only target qualitative properties about input data usage, e.g., only addressing whether an input variable is used or not [22, 23].

In this work, we present a quantitative static analysis for determining the impact of inputs on the program computations. Our approach is based on abstract interpretation, which provides a sound over-approximation of the impact of inputs. We demonstrate our prototype tool in the S2N-BIGNUM library for cryptographic systems to verify the absence of timing side-channels.

**Quantitative Static Timing Analysis**  
Denis Mazzucato<sup>[0000-0002-3613-2035]</sup>, Marco Campion<sup>[0000-0002-1099-3494]</sup>, and Caterina Urban<sup>[0000-0002-8127-9642]</sup>  
INRIA & ENS | PSL,  
`{denis.mazzucato,marco.campion,caterina.urban}@inria.fr`

**Abstract.** Programming errors in software applications can often be difficult to detect, as they may appear without clear indications of failure. One such example is when certain input variables have an unexpected impact on the program's behavior. As an indicator of the program's runtime behavior, this work studies the impact of input variables on the number of loop iterations of a program. Such information is valuable for debugging, optimizing performance, and analyzing security vulnerabilities such as in side-channel attacks where execution times can be exploited. To address this issue, we propose a sound static analysis based on abstract interpretation to quantify the impact of each input variable on the global number of iterations. Our approach combines a dependency analysis with a global loop bound analysis to derive an over-approximation of the impact quantity. We demonstrate our prototype tool in the S2N-BIGNUM library for cryptographic systems to verify the absence of timing side-channels.

**1 Introduction**  
Detecting programming errors that yield a plausible yet incorrect or unsafe behavior is a challenging task. Particularly when these errors do not lead to obvious failures, like program crashes, but may degrade performance or introduce security vulnerabilities. One source of such errors may arise from the unexpected impact of input variables on the program's runtime. For instance, in the context of security, an unexpected impact of input variables on the program's runtime could reveal sensitive information [45], leading to potential security threats. Even in cryptography, where programs are mathematically robust, vulnerabilities to timing attacks persist, depending on the implementation choices and design. Kocher [21] demonstrated that widely used public key cryptographic algorithms, like RSA, are vulnerable to timing attacks, and may leak information about the secret key. The value of such attacks lies in their simplicity; attackers do not need

Quantitative Input Usage Static Analysis  
Denis Mazzucato, Marco Campion, and Caterina Urban  
NFM 2024

- Quantitative Framework
- 3 different Quantifiers
- IMPATTO
- Impact of a few use cases

Quantitative Static Timing Analysis  
Denis Mazzucato, Marco Campion,  
and Caterina Urban  
SAS 2024

- Intensional Property
- Optimized tool TIMESEC
- AWS s2n-bignum
- SVComp
- Artifact

# Contributions

An Abstract Interpretation Framework for Input Data Usage  
Caterina Urban and Peter Müller  
ESOP 2018

Automation of Quantitative Information-Flow Analysis  
Boris Köpf and Andrey Rybalchenko  
SFM 2013

Perfectly Parallel Fairness Certification of Neural Networks  
Caterina Urban et al.  
OOPSLA 2020

Reduced Products of Abstract Domains  
for Fairness Certification of Neural Networks  
Denis Mazzucato and Caterina Urban  
SAS 2021

Reduced Products of Abstract Domains  
for Fairness Certification of Neural Networks

Denis Mazzucato [0000-0002-3613-2035] and Caterina Urban [0000-0002-8127-9642]  
Inria & École Normale Supérieure | Université PSL  
[{denis.mazzucato,caterina.urban}@inria.fr](mailto:{denis.mazzucato,caterina.urban}@inria.fr)

**Abstract.** We present LIBRA, an open-source abstract interpretation-based static analyzer for certifying fairness of ReLU neural network classifiers for tabular data. LIBRA combines a sound forward pre-analysis with an exact backward analysis that leverages the polyhedral abstract domains to provide definite fairness certificates when possible, and to otherwise quantify and describe the biased input space regions. The analysis is configurable in terms of scalability and precision. We equipped LIBRA with new abstract domains to use in the pre-analysis, including a generic reduced product domain construction, as well as search heuristics to find the best analysis configuration. We additionally set up the backward analysis to allow further parallelization. Our experimental evaluation demonstrates the effectiveness of the approach on neural networks trained on a popular dataset in the fairness literature.

**Keywords:** Fairness · Neural Networks · Reduced Abstract Domain Products · Abstract Interpretation · Static Analysis



## 1 Introduction

Nowadays, machine learning software has an ever increasing societal impact by assisting or even automating decision making in fields such as social welfare, criminal justice, and even health care. At the same time, a number of recent cases have shown that such software may reproduce, or even reinforce, bias directly or indirectly present in the training data [3,16,17,23]. In April 2021, the European Commission proposed a first legal framework on machine learning software – the Artificial Intelligence Act [10] – which imposes strict requirements to minimize the risk of discriminatory outcomes. In this context, methods and tools for certifying fairness or otherwise detecting bias are extremely valuable.

## Quantitative Input Usage Static Analysis

Denis Mazzucato [0000-0002-3613-2035], Marco Campion [0000-0002-1099-3494],  
and Caterina Urban [0000-0002-8127-9642]  
INRIA & ENS | PSL  
[{denis.mazzucato,marco.campion,caterina.urban}@inria.fr](mailto:{denis.mazzucato,marco.campion,caterina.urban}@inria.fr)

**Abstract.** Programming errors in software applications may produce plausible yet erroneous results, without providing a clear indication of failure. This happens, for instance, when an input to the program leads to an disproportionate output response. To address this issue, we propose a novel quantitative static analysis for determining the impact of inputs on the program computations, parameterized in the definition of impact. This static analysis employs an underlying abstract backward analyzer and computes a sound over-approximation of the impact of program inputs, providing valuable insights into how the analyzed program handles them. We implement a proof-of-concept static analyzer to demonstrate potential applications.

## 1 Introduction

Disastrous outcomes may result from programming errors in safety-critical settings, especially when they do not result in software failures but instead produce a plausible yet erroneous outcome. Such bugs are hard to spot since they provide no indication that something went wrong. A potential source of such errors is when an input variable has disproportionate impact on the program computations compared to the developers' expectations. A notable example is the Reinhart and Rogoff article "Growth in a Time of Debt" [19], which was heavily cited to justify austerity measures around the world in the following years, and was later discovered to be flawed [13]. Notably, one of the several programming and methodological errors discovered in the article is the incorrect usage of the input value relative to Norway's economic growth in 1964, compromising the authors' conclusion. Hence, it is important to employ techniques that enhance the confidence in the usage of input variables.

In this direction, Barowy et al. [2] proposed a stochastic approach specific for spreadsheet applications. Such approach is able to estimate the impact of input cells. However, the lack of mathematical guarantees precludes the employment of such technique in safety-critical contexts. On the other hand, existing formal methods-based approaches only target qualitative properties about input data usage, e.g., only addressing whether an input variable is used or not [22, 23].

In this work, we present a quantitative static analysis that can discriminate between input variables based on their impact.

## Quantitative Static Timing Analysis

Denis Mazzucato [0000-0002-3613-2035], Marco Campion [0000-0002-1099-3494],  
and Caterina Urban [0000-0002-8127-9642]  
INRIA & ENS | PSL  
[{denis.mazzucato,marco.campion,caterina.urban}@inria.fr](mailto:{denis.mazzucato,marco.campion,caterina.urban}@inria.fr)

**Abstract.** Programming errors in software applications can often be difficult to detect, as they may appear without clear indications of failure. One such example is when certain input variables have an unexpected impact on the program's behavior. As an indicator of the program's runtime behavior, this work studies the impact of input variables on the number of loop iterations of a program. Such information is valuable for debugging, optimizing performance, and identifying security vulnerabilities such as in side-channel attacks where execution times can be exploited. To address this issue, we propose a sound static analysis based on abstract interpretation to quantify the impact of each input variable on the global number of iterations. Our approach combines a dependency analysis with a global loop bound analysis to derive an over-approximation of the impact quantity. We demonstrate our prototype tool in the S2N-BIGNUM library for cryptographic systems to verify the absence of timing side-channels.



## 1 Introduction

Detecting programming errors that yield a plausible yet incorrect or unsafe behavior is a challenging task. Particularly when these errors do not lead to obvious failures, like program crashes, but may degrade performance or introduce security vulnerabilities. One source of such errors may arise from the unexpected impact of input variables on the program's runtime. For instance, in the context of security, an unexpected impact of input variables on the program's runtime could reveal sensitive information [45], leading to potential security threats. Even in cryptography, where programs are mathematically robust, vulnerabilities to timing attacks persist, depending on the implementation choices and design. Kocher [21] demonstrated that widely used public key cryptographic algorithms, like RSA, are vulnerable to timing attacks, and may leak information about the secret key. The value of such attacks lies in their simplicity; attackers do not need

## Quantitative Input Usage Static Analysis

Denis Mazzucato, Marco Campion, and Caterina Urban  
NFM 2024

- Quantitative Framework
- 3 different Quantifiers
- IMPATTO
- Impact of a few use cases

## Quantitative Static Timing Analysis

Denis Mazzucato, Marco Campion,  
and Caterina Urban  
SAS 2024

- Intensional Property
- Optimized tool TIMESEC
- AWS s2n-bignum
- SVComp
- Artifact

# Contributions

An Abstract Interpretation Framework for Input Data Usage  
Caterina Urban and Peter Müller  
ESOP 2018

Automation of Quantitative Information-Flow Analysis  
Boris Köpf and Andrey Rybalchenko  
SFM 2013

Perfectly Parallel Fairness Certification of Neural Networks  
Caterina Urban et al.  
OOPSLA 2020

Reduced Products of Abstract Domains  
for Fairness Certification of Neural Networks  
Denis Mazzucato and Caterina Urban  
SAS 2021

## • Reduced Product

**Quantitative Input Usage Static Analysis**  
Denis Mazzucato<sup>[0000-0002-3613-2035]</sup>, Marco Campion<sup>[0000-0002-1099-3494]</sup>, and Caterina Urban<sup>[0000-0002-8127-9642]</sup>  
INRIA & ENS | PSL,  
`{denis.mazzucato,marco.campion,caterina.urban}@inria.fr`

**Abstract.** Programming errors in software applications may produce plausible yet erroneous results, without providing a clear indication of failure. This happens, for instance, when an input variable has an disproportionate impact on the program's behavior. To address this issue, we propose a novel quantitative static analysis for determining the impact of inputs on the program computations, parameterized in the definition of impact. This static analysis employs an underlying abstract backward analyzer and computes a sound over-approximation of the impact of program inputs, providing valuable insights into how the analyzed program handles them. We implement a proof-of-concept static analyzer to demonstrate potential applications.

**1 Introduction**  
Disastrous outcomes may result from programming errors in safety-critical settings, especially when they do not result in software failures but instead produce a plausible yet erroneous outcome. Such bugs are hard to spot since they provide no indication that something went wrong. A potential source of such errors is when an input variable has disproportionate impact on the program computations compared to the developers' expectations. A notable example is the Reinhart and Rogoff article "Growth in a Time of Debt" [19], which was heavily cited to justify austerity measures around the world in the following years, and was later discovered to be flawed [13]. Notably, one of the several programming and methodological errors discovered in the article is the incorrect usage of the input value relative to Norway's economic growth in 1964, compromising the authors' conclusion. Hence, it is important to employ techniques that enhance the confidence in the usage of input variables.

In this direction, Barowy et al. [2] proposed a stochastic approach specific for spreadsheet applications. Such approach is able to estimate the impact of input cells. However, the lack of mathematical guarantees precludes the employment of such technique in safety-critical contexts. On the other hand, existing formal methods-based approaches only target qualitative properties about input data usage, e.g., only addressing whether an input variable is used or not [22, 23].

In this work, we present a quantitative static analysis for determining the impact of inputs on the program computations, parameterized in the definition of impact. We propose a novel quantitative static analysis for determining the impact of inputs on the program computations, parameterized in the definition of impact. This static analysis employs an underlying abstract backward analyzer and computes a sound over-approximation of the impact of program inputs, providing valuable insights into how the analyzed program handles them. We implement a proof-of-concept static analyzer to demonstrate potential applications.

**Quantitative Static Timing Analysis**  
Denis Mazzucato<sup>[0000-0002-3613-2035]</sup>, Marco Campion<sup>[0000-0002-1099-3494]</sup>, and Caterina Urban<sup>[0000-0002-8127-9642]</sup>  
INRIA & ENS | PSL,  
`{denis.mazzucato,marco.campion,caterina.urban}@inria.fr`

**Abstract.** Programming errors in software applications can often be difficult to detect, as they may appear without clear indications of failure. One such example is when certain input variables have an unexpected impact on the program's behavior. As an indicator of the program's runtime behavior, this work studies the impact of input variables on the number of loop iterations of a program. Such information is valuable for debugging, optimizing performance, and identifying security vulnerabilities such as in side-channel attacks where execution times can be exploited. To address this issue, we propose a sound static analysis based on abstract interpretation to quantify the impact of each input variable on the global number of iterations. Our approach combines a dependency analysis with a global loop bound analysis to derive an over-approximation of the impact quantity. We demonstrate our prototype tool in the S2N-BIGNUM library for cryptographic systems to verify the absence of timing side-channels.

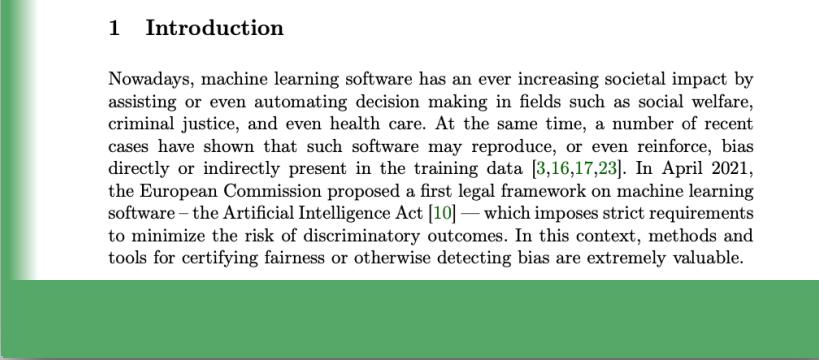
**1 Introduction**  
Detecting programming errors that yield a plausible yet incorrect or unsafe behavior is a challenging task. Particularly when these errors do not lead to obvious failures, like program crashes, but may degrade performance or introduce security vulnerabilities. One source of such errors may arise from the unexpected impact of input variables on the program's runtime. For instance, in the context of security, an unexpected impact of input variables on the program's runtime could reveal sensitive information [45], leading to potential security threats. Even in cryptography, where programs are mathematically robust, vulnerabilities to timing attacks persist, depending on the implementation choices and design. Kocher [21] demonstrated that widely used public key cryptographic algorithms, like RSA, are vulnerable to timing attacks, and may leak information about the secret key. The value of such attacks lies in their simplicity; attackers do not need

Quantitative Input Usage Static Analysis  
Denis Mazzucato, Marco Campion, and Caterina Urban  
NFM 2024

- Quantitative Framework
- 3 different Quantifiers
- IMPATTO
- Impact of a few use cases

Quantitative Static Timing Analysis  
Denis Mazzucato, Marco Campion, and Caterina Urban  
SAS 2024

- Intensional Property
- Optimized tool TIMESEC
- AWS s2n-bignum
- SVComp
- Artifact



# Contributions

An Abstract Interpretation Framework for Input Data Usage  
Caterina Urban and Peter Müller  
ESOP 2018

Automation of Quantitative Information-Flow Analysis  
Boris Köpf and Andrey Rybalchenko  
SFM 2013

Perfectly Parallel Fairness Certification of Neural Networks  
Caterina Urban et al.  
OOPSLA 2020

Reduced Products of Abstract Domains  
for Fairness Certification of Neural Networks  
Denis Mazzucato and Caterina Urban  
SAS 2021

- Reduced Product
- Optimized Parallel Scheduler

**Quantitative Input Usage Static Analysis**  
Denis Mazzucato<sup>[0000-0002-3613-2035]</sup>, Marco Campion<sup>[0000-0002-1099-3494]</sup>, and Caterina Urban<sup>[0000-0002-8127-9642]</sup>  
INRIA & ENS | PSL,  
`{denis.mazzucato,marco.campion,caterina.urban}@inria.fr`

**Abstract.** Programming errors in software applications may produce plausible yet erroneous results, without providing a clear indication of failure. This happens, for instance, when an input variable has an disproportionate impact on the program's behavior. To address this issue, we propose a novel quantitative static analysis for determining the impact of inputs on the program computations, parameterized in the definition of impact. This static analysis employs an underlying abstract backward analyzer and computes a sound over-approximation of the impact of program inputs, providing valuable insights into how the analyzed program handles them. We implement a proof-of-concept static analyzer to demonstrate potential applications.

**1 Introduction**  
Disastrous outcomes may result from programming errors in safety-critical settings, especially when they do not result in software failures but instead produce a plausible yet erroneous outcome. Such bugs are hard to spot since they provide no indication that something went wrong. A potential source of such errors is when an input variable has disproportionate impact on the program computations compared to the developers' expectations. A notable example is the Reinhart and Rogoff article "Growth in a Time of Debt" [19], which was heavily cited to justify austerity measures around the world in the following years, and was later discovered to be flawed [13]. Notably, one of the several programming and methodological errors discovered in the article is the incorrect usage of the input value relative to Norway's economic growth in 1964, compromising the authors' conclusion. Hence, it is important to employ techniques that enhance the confidence in the usage of input variables.

In this direction, Barowy et al. [2] proposed a stochastic approach specific for spreadsheet applications. Such approach is able to estimate the impact of input cells. However, the lack of mathematical guarantees precludes the employment of such technique in safety-critical contexts. On the other hand, existing formal methods-based approaches only target qualitative properties about input data usage, e.g., only addressing whether an input variable is used or not [22, 23].

In this work, we present a quantitative static analysis for determining the impact of inputs on the program computations, parameterized in the definition of impact. We propose a novel quantitative static analysis for determining the impact of inputs on the program computations, parameterized in the definition of impact. This static analysis employs an underlying abstract backward analyzer and computes a sound over-approximation of the impact of program inputs, providing valuable insights into how the analyzed program handles them. We implement a proof-of-concept static analyzer to demonstrate potential applications.

**Quantitative Static Timing Analysis**  
Denis Mazzucato<sup>[0000-0002-3613-2035]</sup>, Marco Campion<sup>[0000-0002-1099-3494]</sup>, and Caterina Urban<sup>[0000-0002-8127-9642]</sup>  
INRIA & ENS | PSL,  
`{denis.mazzucato,marco.campion,caterina.urban}@inria.fr`

**Abstract.** Programming errors in software applications can often be difficult to detect, as they may appear without clear indications of failure. One such example is when certain input variables have an unexpected impact on the program's behavior. As an indicator of the program's runtime behavior, this work studies the impact of input variables on the number of loop iterations of a program. Such information is valuable for debugging, optimizing performance, and identifying security vulnerabilities such as in side-channel attacks where execution times can be exploited. To address this issue, we propose a sound static analysis based on abstract interpretation to quantify the impact of each input variable on the global number of iterations. Our approach combines a dependency analysis with a global loop bound analysis to derive an over-approximation of the impact quantity. We demonstrate our prototype tool in the S2N-BIGNUM library for cryptographic systems to verify the absence of timing side-channels.

**1 Introduction**  
Detecting programming errors that yield a plausible yet incorrect or unsafe behavior is a challenging task. Particularly when these errors do not lead to obvious failures, like program crashes, but may degrade performance or introduce security vulnerabilities. One source of such errors may arise from the unexpected impact of input variables on the program's runtime. For instance, in the context of security, an unexpected impact of input variables on the program's runtime could reveal sensitive information [45], leading to potential security threats. Even in cryptography, where programs are mathematically robust, vulnerabilities to timing attacks persist, depending on the implementation choices and design. Kocher [21] demonstrated that widely used public key cryptographic algorithms, like RSA, are vulnerable to timing attacks, and may leak information about the secret key. The value of such attacks lies in their simplicity; attackers do not need

Quantitative Input Usage Static Analysis  
Denis Mazzucato, Marco Campion, and Caterina Urban  
NFM 2024

- Quantitative Framework
- 3 different Quantifiers
- IMPATTO
- Impact of a few use cases

Quantitative Static Timing Analysis  
Denis Mazzucato, Marco Campion, and Caterina Urban  
SAS 2024

- Intensional Property
- Optimized tool TIMESEC
- AWS s2n-bignum
- SVComp
- Artifact

**Reduced Products of Abstract Domains for Fairness Certification of Neural Networks**  
Denis Mazzucato and Caterina Urban  
SAS 2021

**Abstract.** We present LIBRA, an open-source abstract interpretation-based static analyzer for certifying fairness of ReLU neural network classifiers for tabular data. LIBRA combines a sound forward pre-analysis with an exact backward analysis that leverages the polyhedral abstract domains to provide definite fairness certificates when possible, and to otherwise quantify and describe the biased input space regions. The analysis is configurable in terms of scalability and precision. We equipped LIBRA with new abstract domains to use in the pre-analysis, including a generic reduced product domain construction, as well as search heuristics to find the best analysis configuration. We additionally set up the backward analysis to allow further parallelization. Our experimental evaluation demonstrates the effectiveness of the approach on neural networks trained on a popular dataset in the fairness literature.

**Keywords:** Fairness · Neural Networks · Reduced Abstract Domain Products · Abstract Interpretation · Static Analysis

**1 Introduction**  
Nowadays, machine learning software has an ever increasing societal impact by assisting or even automating decision making in fields such as social welfare, criminal justice, and even health care. At the same time, a number of recent cases have shown that such software may reproduce, or even reinforce, bias directly or indirectly present in the training data [3,16,17,23]. In April 2021, the European Commission proposed a first legal framework on machine learning software – the Artificial Intelligence Act [10] – which imposes strict requirements to minimize the risk of discriminatory outcomes. In this context, methods and tools for certifying fairness or otherwise detecting bias are extremely valuable.

# Contributions

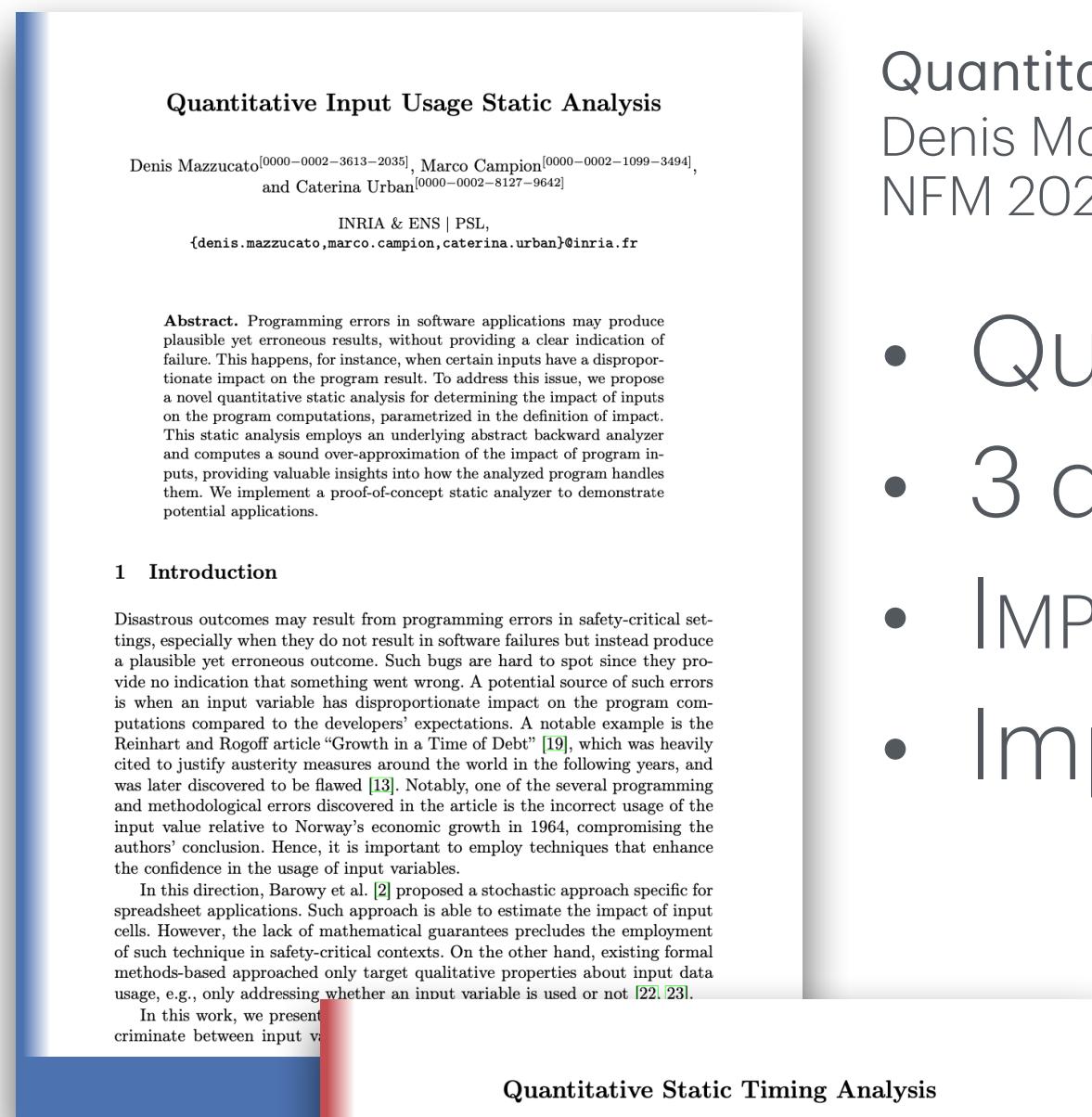
An Abstract Interpretation Framework for Input Data Usage  
Caterina Urban and Peter Müller  
ESOP 2018

Automation of Quantitative Information-Flow Analysis  
Boris Köpf and Andrey Rybalchenko  
SFM 2013

Perfectly Parallel Fairness Certification of Neural Networks  
Caterina Urban et al.  
OOPSLA 2020

Reduced Products of Abstract Domains  
for Fairness Certification of Neural Networks  
Denis Mazzucato and Caterina Urban  
SAS 2021

- Reduced Product
- Optimized Parallel Scheduler
- Performance-vs-Scalability

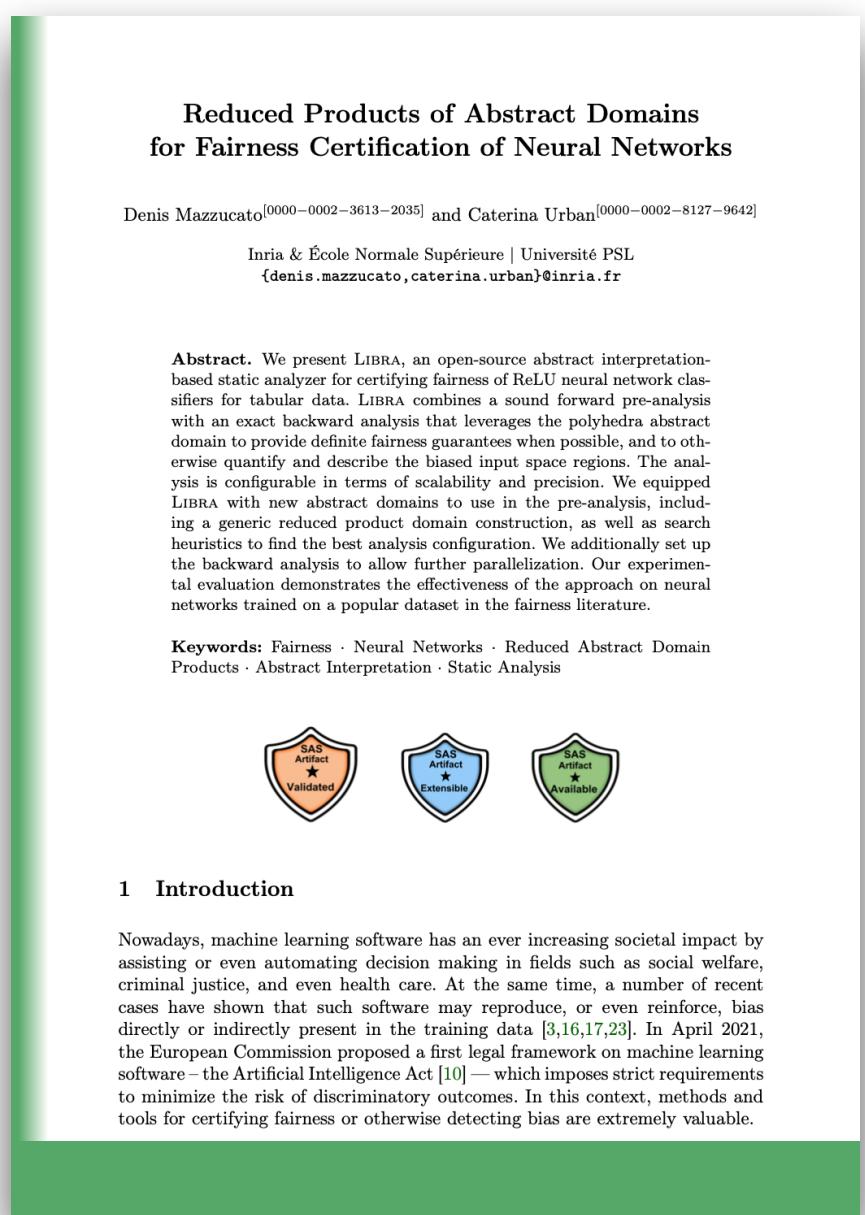


Quantitative Input Usage Static Analysis  
Denis Mazzucato, Marco Campion, and Caterina Urban  
NFM 2024

- Quantitative Framework
- 3 different Quantifiers
- IMPATTO
- Impact of a few use cases

Quantitative Static Timing Analysis  
Denis Mazzucato, Marco Campion,  
and Caterina Urban  
SAS 2024

- Intensional Property
- Optimized tool TIMESEC
- AWS s2n-bignum
- SVComp
- Artifact



# Contributions

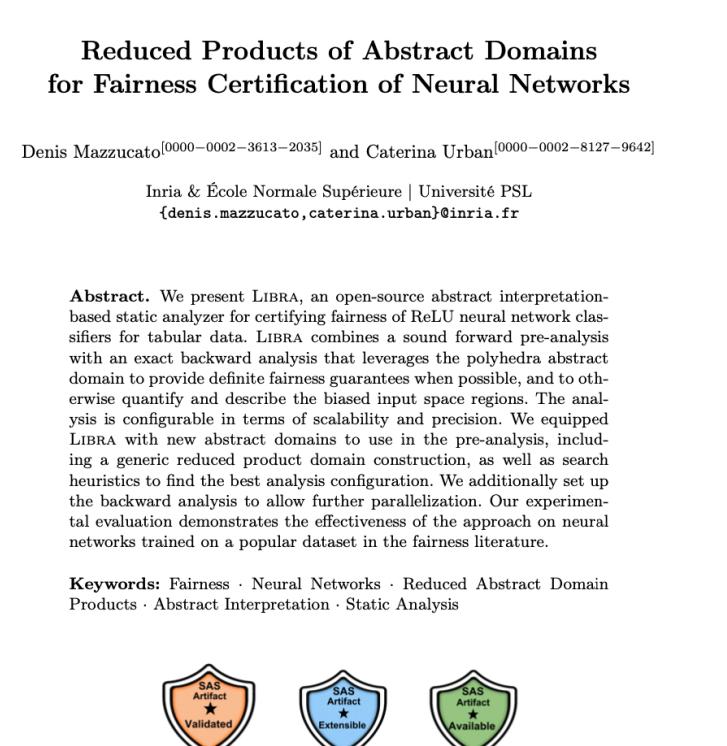
An Abstract Interpretation Framework for Input Data Usage  
Caterina Urban and Peter Müller  
ESOP 2018

Automation of Quantitative Information-Flow Analysis  
Boris Köpf and Andrey Rybalchenko  
SFM 2013

Perfectly Parallel Fairness Certification of Neural Networks  
Caterina Urban et al.  
OOPSLA 2020

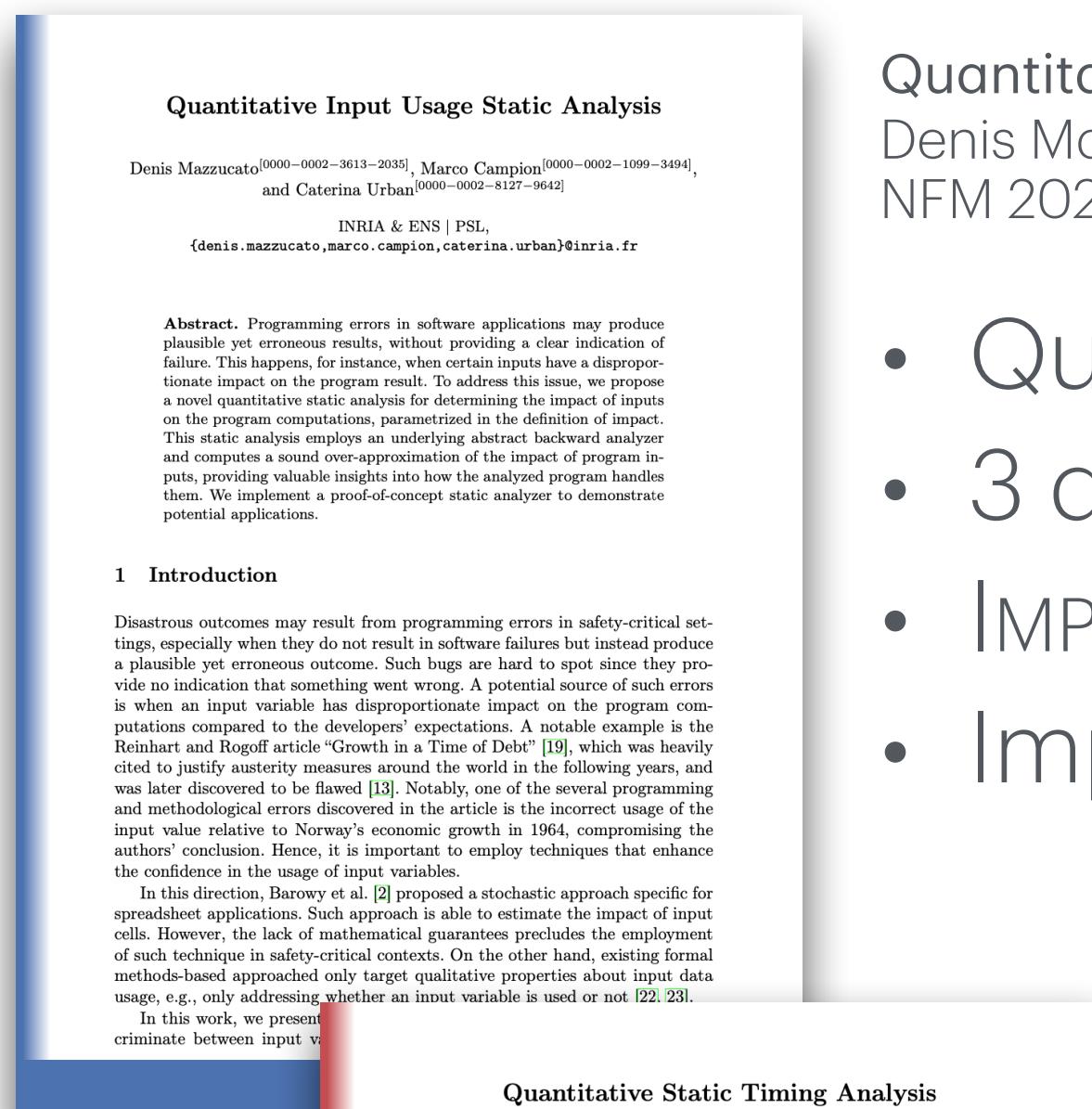
Reduced Products of Abstract Domains  
for Fairness Certification of Neural Networks  
Denis Mazzucato and Caterina Urban  
SAS 2021

- Reduced Product
- Optimized Parallel Scheduler
- Performance-vs-Scalability
- Artifact 



## 1 Introduction

Nowadays, machine learning software has an ever increasing societal impact by assisting or even automating decision making in fields such as social welfare, criminal justice, and even health care. At the same time, a number of recent cases have shown that such software may reproduce, or even reinforce, bias directly or indirectly present in the training data [3,16,17,23]. In April 2021, the European Commission proposed a first legal framework on machine learning software – the Artificial Intelligence Act [10] – which imposes strict requirements to minimize the risk of discriminatory outcomes. In this context, methods and tools for certifying fairness or otherwise detecting bias are extremely valuable.



## Quantitative Input Usage Static Analysis

Denis Mazzucato<sup>[0000-0002-3613-2035]</sup>, Marco Campion<sup>[0000-0002-1099-3494]</sup>,  
and Caterina Urban<sup>[0000-0002-8127-9642]</sup>  
INRIA & ENS | PSL,  
`{denis.mazzucato,marco.campion,caterina.urban}@inria.fr`

**Abstract.** Programming errors in software applications may produce plausible yet erroneous results, without providing a clear indication of failure. This happens, for instance, when an input variable has an disproportionate impact on the program's behavior. To address this issue, we propose a novel quantitative static analysis for determining the impact of inputs on the program computations, parameterized in the definition of impact. This static analysis employs an underlying abstract backward analyzer and computes a sound over-approximation of the impact of program inputs, providing valuable insights into how the analyzed program handles them. We implement a proof-of-concept static analyzer to demonstrate potential applications.

## 1 Introduction

Disastrous outcomes may result from programming errors in safety-critical settings, especially when they do not result in software failures but instead produce a plausible yet erroneous outcome. Such bugs are hard to spot since they provide no indication that something went wrong. A potential source of such errors is when an input variable has disproportionate impact on the program computations compared to the developers' expectations. A notable example is the Reinhart and Rogoff article "Growth in a Time of Debt" [19], which was heavily cited to justify austerity measures around the world in the following years, and was later discovered to be flawed [13]. Notably, one of the several programming and methodological errors discovered in the article is the incorrect usage of the input value relative to Norway's economic growth in 1964, compromising the authors' conclusion. Hence, it is important to employ techniques that enhance the confidence in the usage of input variables.

In this direction, Barowy et al. [2] proposed a stochastic approach specific for spreadsheet applications. Such approach is able to estimate the impact of input cells. However, the lack of mathematical guarantees precludes the employment of such technique in safety-critical contexts. On the other hand, existing formal methods-based approaches only target qualitative properties about input data usage, e.g., only addressing whether an input variable is used or not [22, 23].

In this work, we present a quantitative static analysis that can discriminate between input variables based on their impact.

## Quantitative Static Timing Analysis

Denis Mazzucato<sup>[0000-0002-3613-2035]</sup>, Marco Campion<sup>[0000-0002-1099-3494]</sup>,  
and Caterina Urban<sup>[0000-0002-8127-9642]</sup>  
INRIA & ENS | PSL,  
`{denis.mazzucato,marco.campion,caterina.urban}@inria.fr`

**Abstract.** Programming errors in software applications can often be difficult to detect, as they may appear without clear indications of failure. One such example is when certain input variables have an unexpected impact on the program's behavior. As an indicator of the program's runtime behavior, this work studies the impact of input variables on the number of loop iterations of a program. Such information is valuable for debugging, optimizing performance, and identifying security vulnerabilities such as in side-channel attacks where execution times can be exploited. To address this issue, we propose a sound static analysis based on abstract interpretation to quantify the impact of each input variable on the global number of iterations. Our approach combines a dependency analysis with a global loop bound analysis to derive an over-approximation of the impact quantity. We demonstrate our prototype tool in the S2N-BIGNUM library for cryptographic systems to verify the absence of timing side-channels.



## 1 Introduction

Detecting programming errors that yield a plausible yet incorrect or unsafe behavior is a challenging task. Particularly when these errors do not lead to obvious failures, like program crashes, but may degrade performance or introduce security vulnerabilities. One source of such errors may arise from the unexpected impact of input variables on the program's runtime. For instance, in the context of security, an unexpected impact of input variables on the program's runtime could reveal sensitive information [45], leading to potential security threats. Even in cryptography, where programs are mathematically robust, vulnerabilities to timing attacks persist, depending on the implementation choices and design. Kocher [21] demonstrated that widely used public key cryptographic algorithms, like RSA, are vulnerable to timing attacks, and may leak information about the secret key. The value of such attacks lies in their simplicity; attackers do not need

Quantitative Static Timing Analysis  
Denis Mazzucato, Marco Campion,  
and Caterina Urban  
SAS 2024

- Intensional Property
- Optimized tool TIMESEC
- AWS s2n-bignum
- SVComp
- Artifact 

# Contributions

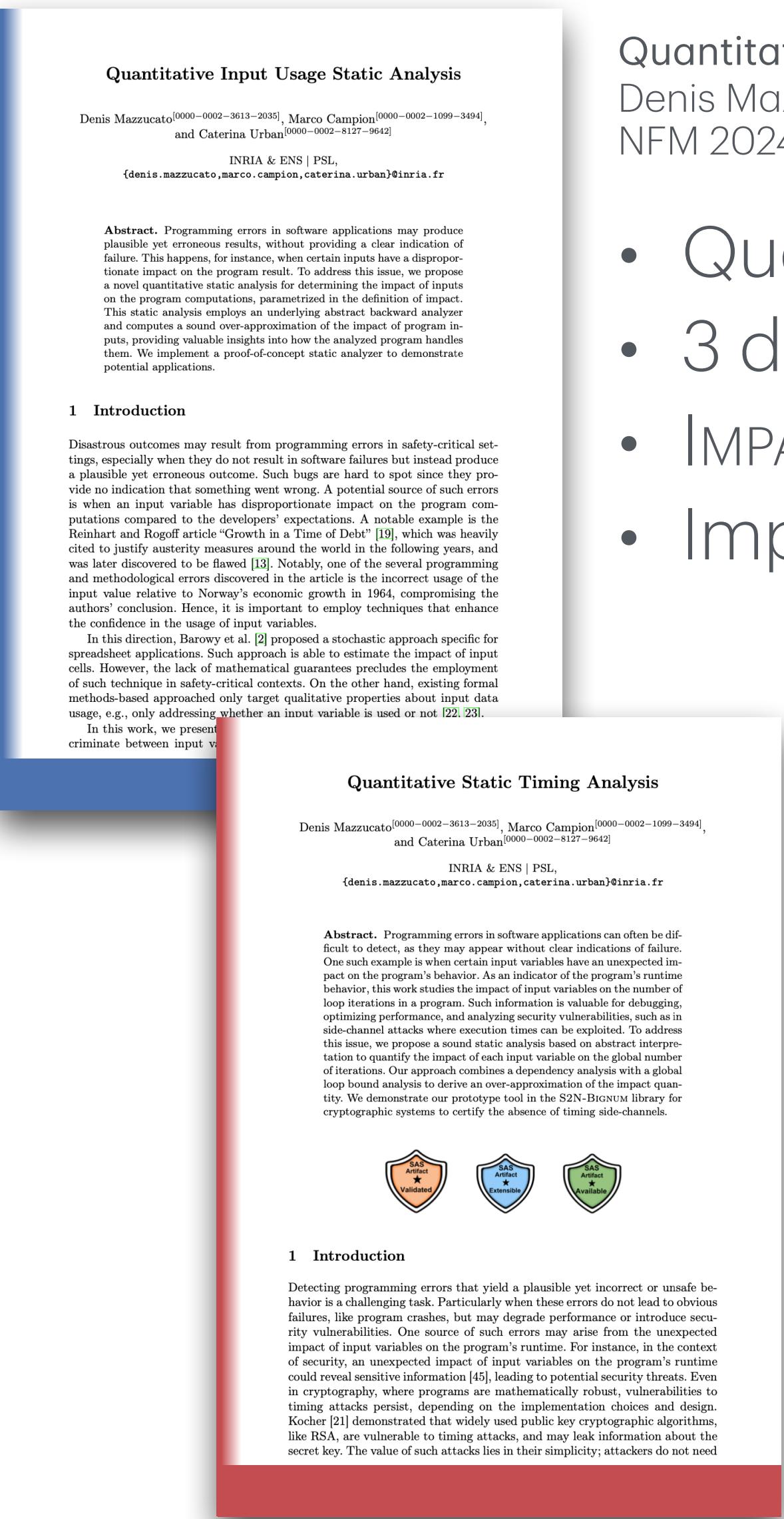
An Abstract Interpretation Framework for Input Data Usage  
Caterina Urban and Peter Müller  
ESOP 2018

Automation of Quantitative Information-Flow Analysis  
Boris Köpf and Andrey Rybalchenko  
SFM 2013

Perfectly Parallel Fairness Certification of Neural Networks  
Caterina Urban et al.  
OOPSLA 2020

Reduced Products of Abstract Domains  
for Fairness Certification of Neural Networks  
Denis Mazzucato and Caterina Urban  
SAS 2021

- Reduced Product
- Optimized Parallel Scheduler
- Performance-vs-Scalability
- Artifact 
- 2 NN Quantifiers



**Quantitative Input Usage Static Analysis**

Denis Mazzucato<sup>[0000-0002-3613-2035]</sup>, Marco Campion<sup>[0000-0002-1099-3494]</sup>, and Caterina Urban<sup>[0000-0002-8127-9642]</sup>  
INRIA & ENS | PSL,  
`{denis.mazzucato,marco.campion,caterina.urban}@inria.fr`

**Abstract.** Programming errors in software applications may produce plausible yet erroneous results, without providing a clear indication of failure. This happens, for instance, when an input variable has a disproportionate impact on the program's behavior. To address this issue, we propose a novel quantitative static analysis for determining the impact of inputs on the program computations, parameterized in the definition of impact. This static analysis employs an underlying abstract backward analyzer and computes a sound over-approximation of the impact of program inputs, providing valuable insights into how the analyzed program handles them. We implement a proof-of-concept static analyzer to demonstrate potential applications.

**1 Introduction**

Disastrous outcomes may result from programming errors in safety-critical settings, especially when they do not result in software failures but instead produce a plausible yet erroneous outcome. Such bugs are hard to spot since they provide no indication that something went wrong. A potential source of such errors is when an input variable has disproportionate impact on the program computations compared to the developers' expectations. A notable example is the Reinhart and Rogoff article "Growth in a Time of Debt" [19], which was heavily cited to justify austerity measures around the world in the following years, and was later discovered to be flawed [13]. Notably, one of the several programming and methodological errors discovered in the article is the incorrect usage of the input value relative to Norway's economic growth in 1964, compromising the authors' conclusion. Hence, it is important to employ techniques that enhance the confidence in the usage of input variables.

In this direction, Barowy et al. [2] proposed a stochastic approach specific for spreadsheet applications. Such approach is able to estimate the impact of input cells. However, the lack of mathematical guarantees precludes the employment of such technique in safety-critical contexts. On the other hand, existing formal methods-based approaches only target qualitative properties about input data usage, e.g., only addressing whether an input variable is used or not [22, 23].

In this work, we present a quantitative static analysis for determining the impact of inputs on the program computations. Our approach is based on abstract interpretation, combining a sound dependency analysis with a global loop bound analysis to derive an over-approximation of the impact quantity. We demonstrate our prototype tool in the S2N-BIGNUM library for cryptographic systems to verify the absence of timing side-channels.

**Quantitative Static Timing Analysis**

Denis Mazzucato<sup>[0000-0002-3613-2035]</sup>, Marco Campion<sup>[0000-0002-1099-3494]</sup>, and Caterina Urban<sup>[0000-0002-8127-9642]</sup>  
INRIA & ENS | PSL,  
`{denis.mazzucato,marco.campion,caterina.urban}@inria.fr`

**Abstract.** Programming errors in software applications can often be difficult to detect, as they may appear without clear indications of failure. One such example is when certain input variables have an unexpected impact on the program's behavior. As an indicator of the program's runtime behavior, this work studies the impact of input variables on the number of loop iterations of a program. Such information is valuable for debugging, optimizing performance, and identifying security vulnerabilities, such as in side-channel attacks where execution times can be exploited. To address this issue, we propose a sound static analysis based on abstract interpretation to quantify the impact of each input variable on the global number of iterations. Our approach combines a dependency analysis with a global loop bound analysis to derive an over-approximation of the impact quantity. We demonstrate our prototype tool in the S2N-BIGNUM library for cryptographic systems to verify the absence of timing side-channels.



**1 Introduction**

Detecting programming errors that yield a plausible yet incorrect or unsafe behavior is a challenging task. Particularly when these errors do not lead to obvious failures, like program crashes, but may degrade performance or introduce security vulnerabilities. One source of such errors may arise from the unexpected impact of input variables on the program's runtime. For instance, in the context of security, an unexpected impact of input variables on the program's runtime could reveal sensitive information [45], leading to potential security threats. Even in cryptography, where programs are mathematically robust, vulnerabilities to timing attacks persist, depending on the implementation choices and design. Kocher [21] demonstrated that widely used public key cryptographic algorithms, like RSA, are vulnerable to timing attacks, and may leak information about the secret key. The value of such attacks lies in their simplicity; attackers do not need

Quantitative Input Usage Static Analysis  
Denis Mazzucato, Marco Campion, and Caterina Urban  
NFM 2024

- Quantitative Framework
- 3 different Quantifiers
- IMPATTO
- Impact of a few use cases

Quantitative Static Timing Analysis  
Denis Mazzucato, Marco Campion,  
and Caterina Urban  
SAS 2024

- Intensional Property
- Optimized tool TIMESEC
- AWS s2n-bignum
- SVComp
- Artifact 

# Contributions

An Abstract Interpretation Framework for Input Data Usage  
Caterina Urban and Peter Müller  
ESOP 2018

Automation of Quantitative Information-Flow Analysis  
Boris Köpf and Andrey Rybalchenko  
SFM 2013

Perfectly Parallel Fairness Certification of Neural Networks  
Caterina Urban et al.  
OOPSLA 2020

Reduced Products of Abstract Domains  
for Fairness Certification of Neural Networks  
Denis Mazzucato and Caterina Urban  
SAS 2021

- Reduced Product
- Optimized Parallel Scheduler
- Performance-vs-Scalability
- Artifact **ze**
- 2 NN Quantifiers
- Quantitative Experiments

**Quantitative Input Usage Static Analysis**

Denis Mazzucato<sup>[0000-0002-3613-2035]</sup>, Marco Campion<sup>[0000-0002-1099-3494]</sup>, and Caterina Urban<sup>[0000-0002-8127-9642]</sup>  
INRIA & ENS | PSL,  
`{denis.mazzucato,marco.campion,caterina.urban}@inria.fr`

**Abstract.** Programming errors in software applications may produce plausible yet erroneous results, without providing a clear indication of failure. This happens, for instance, when an input variable has an disproportionate impact on the program's behavior. To address this issue, we propose a novel quantitative static analysis for determining the impact of inputs on the program computations, parameterized in the definition of impact. This static analysis employs an underlying abstract backward analyzer and computes a sound over-approximation of the impact of program inputs, providing valuable insights into how the analyzed program handles them. We implement a proof-of-concept static analyzer to demonstrate potential applications.

**1 Introduction**

Disastrous outcomes may result from programming errors in safety-critical settings, especially when they do not result in software failures but instead produce a plausible yet erroneous outcome. Such bugs are hard to spot since they provide no indication that something went wrong. A potential source of such errors is when an input variable has disproportionate impact on the program computations compared to the developers' expectations. A notable example is the Reinhart and Rogoff article "Growth in a Time of Debt" [19], which was heavily cited to justify austerity measures around the world in the following years, and was later discovered to be flawed [13]. Notably, one of the several programming and methodological errors discovered in the article is the incorrect usage of the input value relative to Norway's economic growth in 1964, compromising the authors' conclusion. Hence, it is important to employ techniques that enhance the confidence in the usage of input variables.

In this direction, Barowy et al. [2] proposed a stochastic approach specific for spreadsheet applications. Such approach is able to estimate the impact of input cells. However, the lack of mathematical guarantees precludes the employment of such technique in safety-critical contexts. On the other hand, existing formal methods-based approaches only target qualitative properties about input data usage, e.g., only addressing whether an input variable is used or not [22, 23].

In this work, we present a quantitative static analysis for determining the impact of inputs on the program computations. Our approach is based on abstract interpretation, combining a sound dependency analysis with a global loop bound analysis to derive an over-approximation of the impact quantity. We demonstrate our prototype tool in the S2N-BIGNUM library for cryptographic systems to verify the absence of timing side-channels.

**Quantitative Static Timing Analysis**

Denis Mazzucato<sup>[0000-0002-3613-2035]</sup>, Marco Campion<sup>[0000-0002-1099-3494]</sup>, and Caterina Urban<sup>[0000-0002-8127-9642]</sup>  
INRIA & ENS | PSL,  
`{denis.mazzucato,marco.campion,caterina.urban}@inria.fr`

**Abstract.** Programming errors in software applications can often be difficult to detect, as they may appear without clear indications of failure. One such example is when certain input variables have an unexpected impact on the program's behavior. As an indicator of the program's runtime behavior, this work studies the impact of input variables on the number of loop iterations of a program. Such information is valuable for debugging, optimizing performance, and identifying security vulnerabilities, such as in side-channel attacks where execution times can be exploited. To address this issue, we propose a sound static analysis based on abstract interpretation to quantify the impact of each input variable on the global number of iterations. Our approach combines a dependency analysis with a global loop bound analysis to derive an over-approximation of the impact quantity. We demonstrate our prototype tool in the S2N-BIGNUM library for cryptographic systems to verify the absence of timing side-channels.

**1 Introduction**

Detecting programming errors that yield a plausible yet incorrect or unsafe behavior is a challenging task. Particularly when these errors do not lead to obvious failures, like program crashes, but may degrade performance or introduce security vulnerabilities. One source of such errors may arise from the unexpected impact of input variables on the program's runtime. For instance, in the context of security, an unexpected impact of input variables on the program's runtime could reveal sensitive information [45], leading to potential security threats. Even in cryptography, where programs are mathematically robust, vulnerabilities to timing attacks persist, depending on the implementation choices and design. Kocher [21] demonstrated that widely used public key cryptographic algorithms, like RSA, are vulnerable to timing attacks, and may leak information about the secret key. The value of such attacks lies in their simplicity; attackers do not need

Quantitative Input Usage Static Analysis  
Denis Mazzucato, Marco Campion, and Caterina Urban  
NFM 2024

- Quantitative Framework
- 3 different Quantifiers
- IMPATTO
- Impact of a few use cases

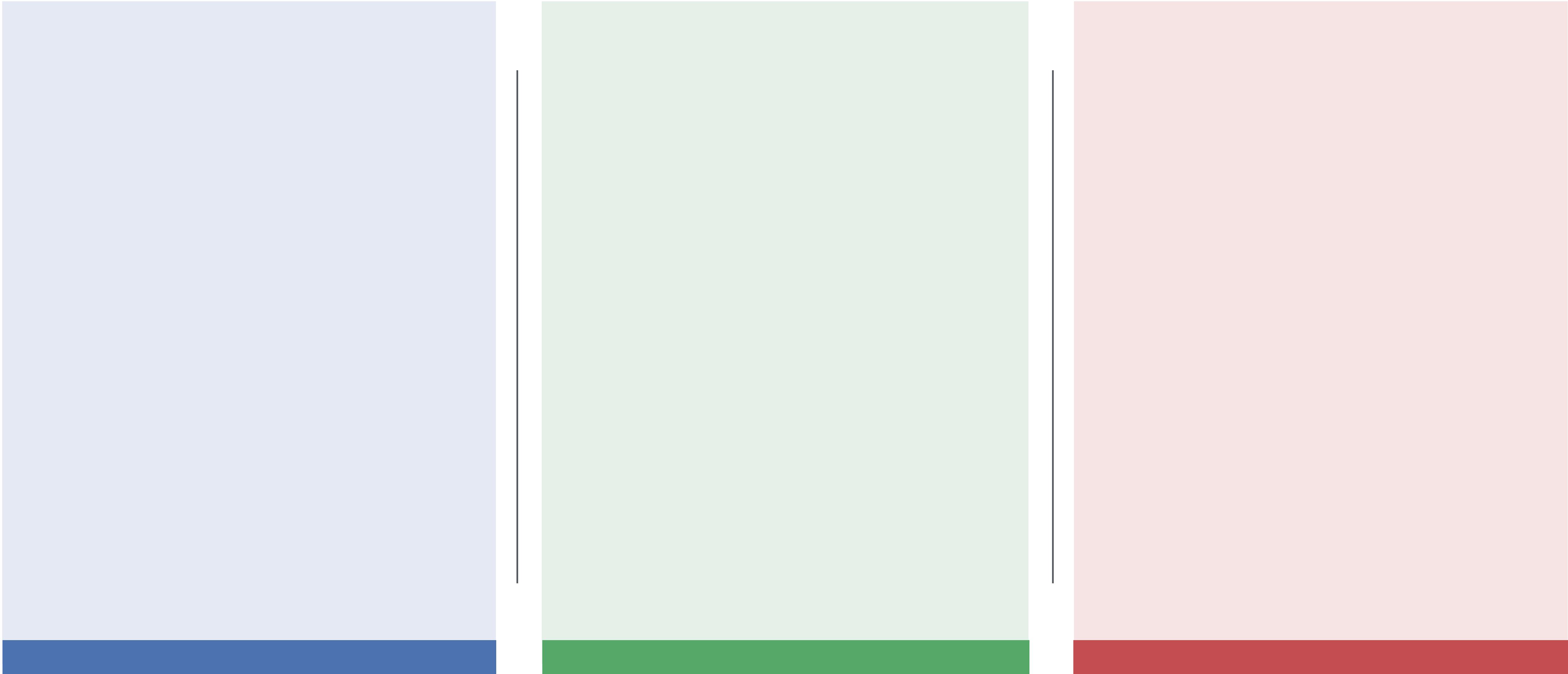
Quantitative Static Timing Analysis  
Denis Mazzucato, Marco Campion,  
and Caterina Urban  
SAS 2024

- Intensional Property
- Optimized tool TIMESEC
- AWS s2n-bignum
- SVComp
- Artifact **ze**



# Future Work

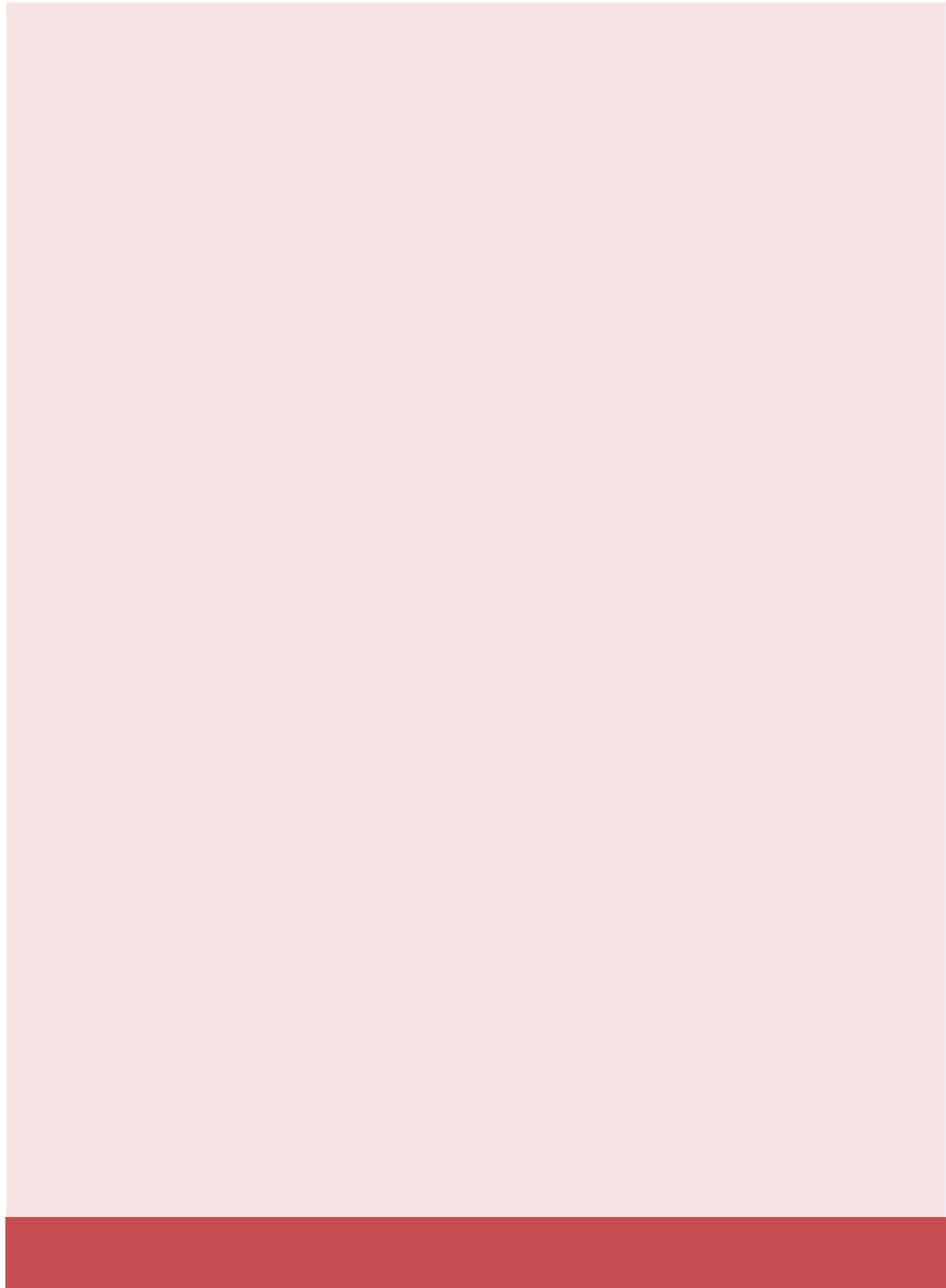
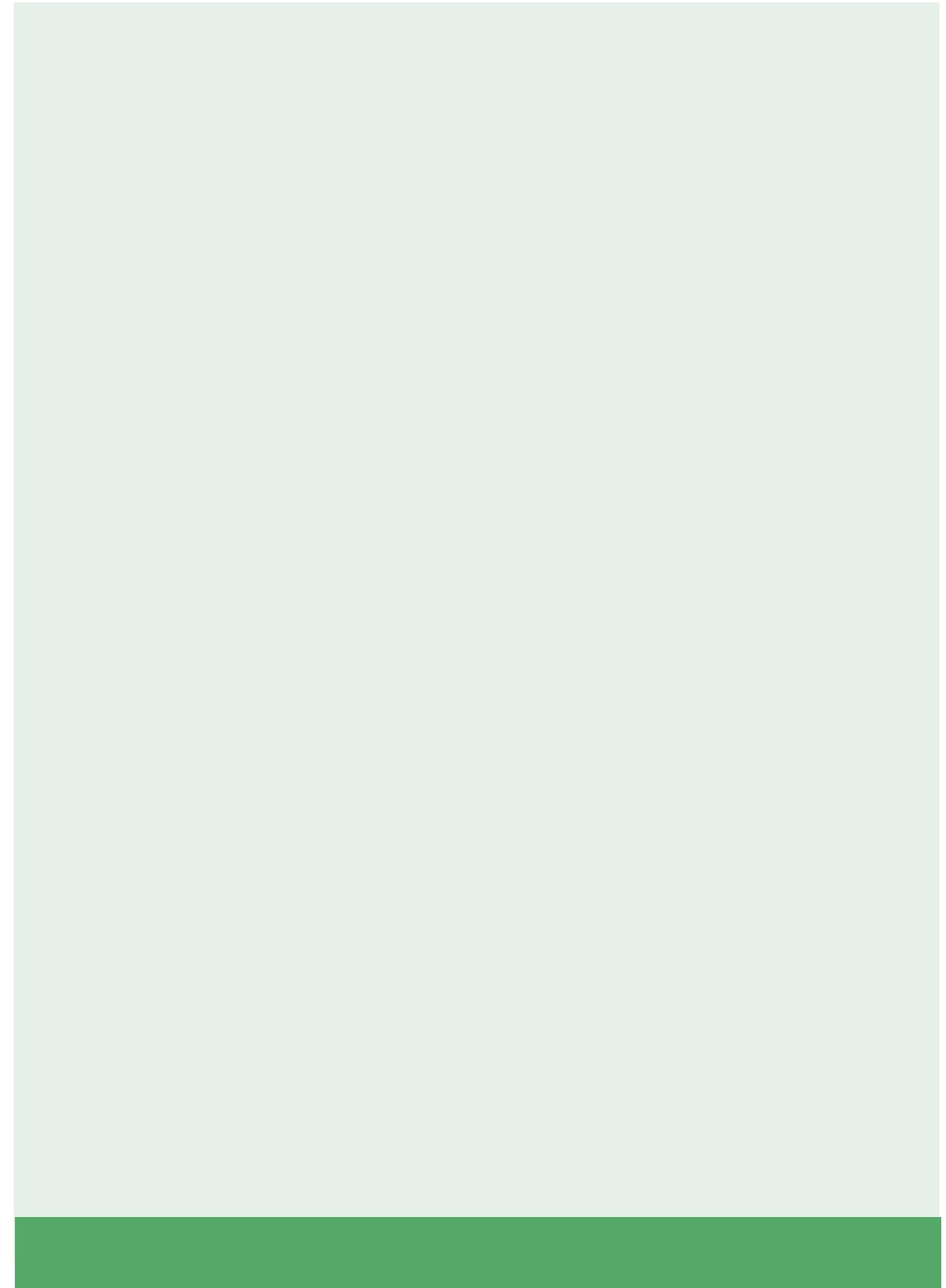
---



# Future Work

---

- Precise Abstractions
- Non-Termination
- Non-Determinism
- Obfuscation
- ...



# Future Work

---

- Precise Abstractions
- Non-Termination
- Non-Determinism
- Obfuscation
- ...

- Neural Network Verification
- Data Science Notebooks
- ...

# Future Work

---

- Precise Abstractions
- Non-Termination
- Non-Determinism
- Obfuscation
- ...

- Neural Network Verification
- Data Science Notebooks
- ...

- Post-Spectre Era
- Other Side-Channel Attacks
- ...

# Static Analysis By Abstract Interpretation for Quantitative Program Properties

Denis Mazzucato – PhD Defense

10 December 2024

## Theoretical Framework

**Quantitative Input Usage Static Analysis**

Denis Mazzucato<sup>[0000-0002-3613-2035]</sup>, Marco Campion<sup>[0000-0002-1099-3494]</sup>, and Caterina Urban<sup>[0000-0002-8127-9642]</sup>

INRIA & ENS | PSL,  
`{denis.mazzucato,marco.campion,caterina.urban}@inria.fr`

**Abstract.** Programming errors in software applications may produce plausible yet erroneous results, without providing a clear indication of failure. This happens, for instance, when certain inputs have a disproportionate impact on the program result. To address this issue, we propose a novel quantitative static analysis for determining the impact of inputs on the program computations, parametrized in the definition of impact. This static analysis employs an underlying abstract backward analyzer and computes a sound over-approximation of the impact of program inputs, providing valuable insights into how the analyzed program handles them. We implement a proof-of-concept static analyzer to demonstrate potential applications.

**1 Introduction**

Disastrous outcomes may result from programming errors in safety-critical settings, especially when they do not result in software failures but instead produce a plausible yet erroneous outcome. Such bugs are hard to spot since they provide no indication that something went wrong. A potential source of such errors is when an input variable has disproportionate impact on the program computations compared to the developers' expectations. A notable example is the Reinhart and Rogoff article "Growth in a Time of Debt" [19], which was heavily cited to justify austerity measures around the world in the following years, and was later discovered to be flawed [13]. Notably, one of the several programming and methodological errors discovered in the article is the incorrect usage of the input value relative to Norway's economic growth in 1964, compromising the authors' conclusion. Hence, it is important to employ techniques that enhance the confidence in the usage of input variables.

In this direction, Barowy et al. [2] proposed a stochastic approach specific for spreadsheet applications. Such approach is able to estimate the impact of input cells. However, the lack of mathematical guarantees precludes the employment of such technique in safety-critical contexts. On the other hand, existing formal methods-based approaches only target qualitative properties about input data usage, e.g., only addressing whether an input variable is used or not [22, 23].

In this work, we present a novel quantitative input usage framework to discriminate between input variables with different impact on the outcome of a

## Extensional Properties

**Reduced Products of Abstract Domains for Fairness Certification of Neural Networks**

Denis Mazzucato<sup>[0000-0002-3613-2035]</sup> and Caterina Urban<sup>[0000-0002-8127-9642]</sup>

Inria & École Normale Supérieure | Université PSL  
`{denis.mazzucato,caterina.urban}@inria.fr`

**Abstract.** We present LIBRA, an open-source abstract interpretation-based static analyzer for certifying fairness of ReLU neural network classifiers for tabular data. LIBRA combines a sound forward pre-analysis with an exact backward analysis that leverages the polyhedra abstract domain to provide definite fairness guarantees when possible, and to otherwise quantify and describe the biased input space regions. The analysis is configurable in terms of scalability and precision. We equipped LIBRA with new abstract domains to use in the pre-analysis, including a generic reduced product domain construction, as well as search heuristics to find the best analysis configuration. We additionally set up the backward analysis to allow further parallelization. Our experimental evaluation demonstrates the effectiveness of the approach on neural networks trained on a popular dataset in the fairness literature.

**Keywords:** Fairness · Neural Networks · Reduced Abstract Domain Products · Abstract Interpretation · Static Analysis



## Intensional Properties

**Quantitative Static Timing Analysis**

Denis Mazzucato<sup>[0000-0002-3613-2035]</sup>, Marco Campion<sup>[0000-0002-1099-3494]</sup>, and Caterina Urban<sup>[0000-0002-8127-9642]</sup>

INRIA & ENS | PSL,  
`{denis.mazzucato,marco.campion,caterina.urban}@inria.fr`

**Abstract.** Programming errors in software applications can often be difficult to detect, as they may appear without clear indications of failure. One such example is when certain input variables have an unexpected impact on the program's behavior. As an indicator of the program's runtime behavior, this work studies the impact of input variables on the number of loop iterations in a program. Such information is valuable for debugging, optimizing performance, and analyzing security vulnerabilities, such as in side-channel attacks where execution times can be exploited. To address this issue, we propose a sound static analysis based on abstract interpretation to quantify the impact of each input variable on the global number of iterations. Our approach combines a dependency analysis with a global loop bound analysis to derive an over-approximation of the impact quantity. We demonstrate our prototype tool in the S2N-BIGNUM library for cryptographic systems to certify the absence of timing side-channels.



# Questions?

Reduced Products of Abstract Domains  
for Fairness Certification of Neural Networks

Denis Mazzucato<sup>[0000-0002-3613-2035]</sup> and Caterina Urban<sup>[0000-0002-8127-9642]</sup>

Inria & École Normale Supérieure | Université PSL  
[{denis.mazzucato,caterina.urban}@inria.fr](mailto:{denis.mazzucato,caterina.urban}@inria.fr)

**Abstract.** We present LIBRA, an open-source abstract interpretation-based static analyzer for certifying fairness of ReLU neural network classifiers for tabular data. LIBRA combines a sound forward pre-analysis with an exact backward analysis that leverages the polyhedra abstract domain to provide definite fairness guarantees when possible, and to otherwise quantify and describe the biased input space regions. The analysis is configurable in terms of scalability and precision. We equipped LIBRA with new abstract domains to use in the pre-analysis, including a generic reduced product domain construction, as well as search heuristics to find the best analysis configuration. We additionally set up the backward analysis to allow further parallelization. Our experimental evaluation demonstrates the effectiveness of the approach on neural networks trained on a popular dataset in the fairness literature.

**Keywords:** Fairness · Neural Networks · Reduced Abstract Domain Products · Abstract Interpretation · Static Analysis



**1 Introduction**

Nowadays, machine learning software has an ever increasing societal impact by assisting or even automating decision making in fields such as social welfare, criminal justice, and even health care. At the same time, a number of recent cases have shown that such software may reproduce, or even reinforce, bias directly or indirectly present in the training data [3,16,17,23]. In April 2021, the European Commission proposed a first legal framework on machine learning software – the Artificial Intelligence Act [10] – which imposes strict requirements to minimize the risk of discriminatory outcomes. In this context, methods and tools for certifying fairness or otherwise detecting bias are extremely valuable.

- Reduced Product
- Optimized Parallel Scheduler
- Performance-vs-Scalability
- Artifact **ze**
- 2 NN Quantifiers
- Quantitative Experiments

Quantitative Input Usage Static Analysis

Denis Mazzucato<sup>[0000-0002-3613-2035]</sup>, Marco Campion<sup>[0000-0002-1099-3494]</sup>, and Caterina Urban<sup>[0000-0002-8127-9642]</sup>

INRIA & ENS | PSL,  
[{denis.mazzucato,marco.campion,caterina.urban}@inria.fr](mailto:{denis.mazzucato,marco.campion,caterina.urban}@inria.fr)

**Abstract.** Programming errors in software applications may produce plausible yet erroneous results, without providing a clear indication of failure. This happens, for instance, when a variable is used in a disproportionate way in a program. To address this issue, we propose a novel quantitative static analysis for determining the impact of inputs on the program computations, parameterized in the definition of impact. This static analysis employs an underlying abstract backward analyzer and computes a sound over-approximation of the impact of program inputs, providing valuable insights into how the analyzed program handles them. We implement a proof-of-concept static analyzer to demonstrate potential applications.

**1 Introduction**

Disastrous outcomes may result from programming errors in safety-critical settings, especially when they do not result in software failures but instead produce a plausible yet erroneous outcome. Such bugs are hard to spot since they provide no indication that something went wrong. A notable source of such errors is when an input variable has disproportionate impact on the program computations compared to the developers' expectations. A notable example is the Reinhart and Rogoff article "Growth in a Time of Debt" [19], which was heavily cited to justify austerity measures around the world in the following years, and was later discovered to be flawed [13]. Notably, one of the several programming and methodological errors discovered in the article is the incorrect usage of the input value relative to Norway's economic growth in 1964, compromising the authors' conclusion. Hence, it is important to employ techniques that enhance the confidence in the usage of input variables.

In this direction, Barowy et al. [2] proposed a stochastic approach specific for spreadsheet applications. Such approach is able to estimate the impact of input cells. However, the lack of mathematical guarantees precludes the employment of such technique in safety-critical contexts. On the other hand, existing formal methods-based approaches only target qualitative properties about input data usage, e.g., only addressing whether an input variable is used or not [22, 23].

In this work, we present a quantitative static analysis that can discriminate between input variables based on their impact on the program's runtime behavior.

**Quantitative Static Timing Analysis**

Denis Mazzucato<sup>[0000-0002-3613-2035]</sup>, Marco Campion<sup>[0000-0002-1099-3494]</sup>, and Caterina Urban<sup>[0000-0002-8127-9642]</sup>

INRIA & ENS | PSL,  
[{denis.mazzucato,marco.campion,caterina.urban}@inria.fr](mailto:{denis.mazzucato,marco.campion,caterina.urban}@inria.fr)

**Abstract.** Programming errors in software applications can often be difficult to detect, as they may appear without clear indications of failure. One such example is when certain input variables have an unexpected impact on the program's behavior. As an indicator of the program's runtime behavior, this work studies the impact of input variables on the number of loop iterations of a program. Such information is valuable for debugging, optimizing performance, and analyzing security vulnerabilities such as in side-channel attacks where execution times can be exploited. To address this issue, we propose a sound static analysis based on abstract interpretation to quantify the impact of each input variable on the global number of iterations. Our approach combines a dependency analysis with a global loop bound analysis to derive an over-approximation of the impact quantity. We demonstrate our prototype tool in the S2N-BIGNUM library for cryptographic systems to verify the absence of timing side-channels.



**1 Introduction**

Detecting programming errors that yield a plausible yet incorrect or unsafe behavior is a challenging task. Particularly when these errors do not lead to obvious failures, like program crashes, but may degrade performance or introduce security vulnerabilities. One source of such errors may arise from the unexpected impact of input variables on the program's runtime. For instance, in the context of security, an unexpected impact of input variables on the program's runtime could reveal sensitive information [45], leading to potential security threats. Even in cryptography, where programs are mathematically robust, vulnerabilities to timing attacks persist, depending on the implementation choices and design. Kocher [21] demonstrated that widely used public key cryptographic algorithms, like RSA, are vulnerable to timing attacks, and may leak information about the secret key. The value of such attacks lies in their simplicity; attackers do not need



Radhia Cousot Award

Quantitative Input Usage Static Analysis  
Denis Mazzucato, Marco Campion, and Caterina Urban  
NFM 2024

- Quantitative Framework
- 3 different Quantifiers
- IMPATTO
- Impact of a few use cases

Quantitative Static Timing Analysis

Denis Mazzucato, Marco Campion,  
and Caterina Urban  
SAS 2024

- Intensional Property
- Optimized tool TIMESEC
- AWS s2n-bignum
- SVComp
- Artifact **ze**