Static Analysis for Data Science

MPRI 2-6: Abstract Interpretation, Application to Verification and Static Analysis

Caterina Urban

December 14th, 2023

Year 2023-2024

Data Science is Everywhere

data is cheap and ubiquitous

web logs

mobile devices

sensors

transactions

data science is revolutionizing industries

retail
- personalized recommendations
- targeted marketing

pharmaceutical
- predictive models
- patient selection

manufacturing
- equipment failure predictions
- internet of things

finance
- predictive models
- customized product offerings

energy
- exploration and discovery
- accident prevention

health care
- personalized treatments
- preventive care

DATA SCIENCE SOFTWARE

data

data preparation

model training

model deployment

predictions

TODAY

NEXT LESSON

Ubiquitous Programming Errors

data science means programming

Excel

R

Python

Julia

programming means programming errors

programming errors that do not cause failures can have **serious consequences**

pharmaceutical

energy

finance

health care

Anomalously Unused Data

The Reinhart-Rogo

	B	C	I	J	K	L	M
2			Real GDP growth				
3			Debt/GDP				
4	Country	Coverage	30 or less	30 to 60	60 to 90	90 or above	30 or less
26			3.7	3.0	3.5	1.7	5.5
27	Minimum		1.6	0.3	1.3	-1.8	0.8
						3.6	13.3
						-2.0	n.a.
						2.4	n.a.
						n.a.	6.3
						n.a.	9.9
						n.a.	7.9

FAQ: Reinhart, Rogoff, and the Excel Error That Changed History

By Peter Coy | April 18, 2013

The Excel Depression

By PAUL KRUGMAN
Published: April 18, 2013 | 470 Comments

In this age of information, math errors can lead to disaster. NASA's Mars Orbiter crashed because engineers forgot to convert to metric measurements; JPMorgan Chase's "London Whale" venture went bad in part because modelers divided by a sum instead of an average. So, did an Excel coding error destroy the economies of the Western world?

Enlarge This Image

The story so far: At the beginning of 2010, two Harvard economists, Carmen Reinhart and Kenneth Rogoff, circulated a paper, "Growth in a Time of Debt," that purported to identify a critical "threshold," a tipping point, for government indebtedness. Once debt exceeds 90 percent of gross domestic product, they claimed, economic growth drops off sharply.

Ms. Reinhart and Mr. Rogoff had credibility thanks to a widely admired earlier book on the history of financial

England Covid-19 Cases Error

SCIENCE | US & WORLD | TECH

Excel spreadsheet error blamed for UK's 16,000 missing coronavirus cases

The case went missing after the spreadsheet hit its filesize limit

By James Vincent | Oct 5, 2020, 9:41am EDT

NEWS

Check for updates

The BMJ

Cite this as: *BMJ* 2020;371:m3891
http://dx.doi.org/10.1136/bmj.m3891
Published: 06 October 2020

Covid-19: Only half of 16 000 patients missed from England's official figures have been contacted

Elisabeth Mahase

Details of nearly 16 000 cases of covid-19 were not transferred to England's NHS Test and Trace service and were missed from official figures because of an error in the process for updating the data.

England's health and social care secretary, Matt Hancock, told the House of Commons on Monday 5 October that after the error was discovered on Friday 2 October "6500 hours of extra contact tracing" had been carried out over the weekend. But as at Monday morning only half (51%) of the people had been reached by contact tracers.

In response, Labour's shadow health secretary, Jonathan Ashworth, said, "Thousands of people are blissfully unaware they have been exposed to covid, spreading this deadly virus at a time when [illegible] and we are in the [illegible] much data and furthermore have issued guidance on validation and risk management for these products if they are to be used in such a safety critical manner."

The error came as the Labour Party's leader, Keir Starmer, said that the prime minister had "lost control" of covid-19, with no clear strategy for beating it. Speaking to the *Observer*, Starmer set out his five point plan for covid-19, which starts with publishing the criteria for local restrictions, as the German government did. Secondly, he said public health messaging should be improved by adding a feature to the NHS covid-19 app so people can search their postcode and find out their local restrictions.

Starmer has also said he would fix the contact tracing system by investing in NHS and university laboratories to expand testing and at the same time put local public health teams in charge of contact [illegible] in their areas. Routine regular testing in high [illegible] high transmission areas would [illegible] in 24 hours.

Example

```
english = bool(input())
math = bool(input())
science = bool(input())
bonus = bool(input())

passing = True
if not english:
    english = False
if not math:
    passing = False or bonus
if not math:
    passing = False or bonus

print(passing)
```

INPUT VARIABLES

ERROR: english SHOULD BE passing

ERROR: math SHOULD BE science

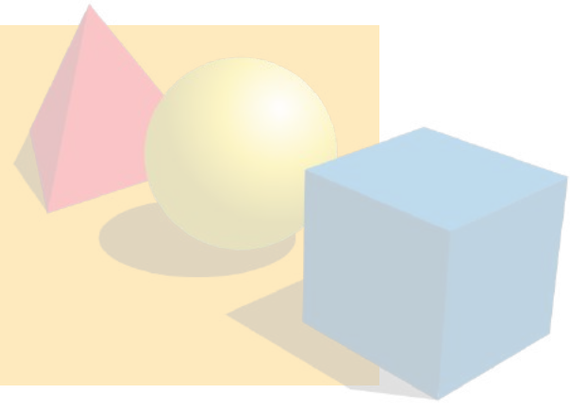
OUTPUT VARIABLES

the input variables english and science are unused

Unused Data Analysis

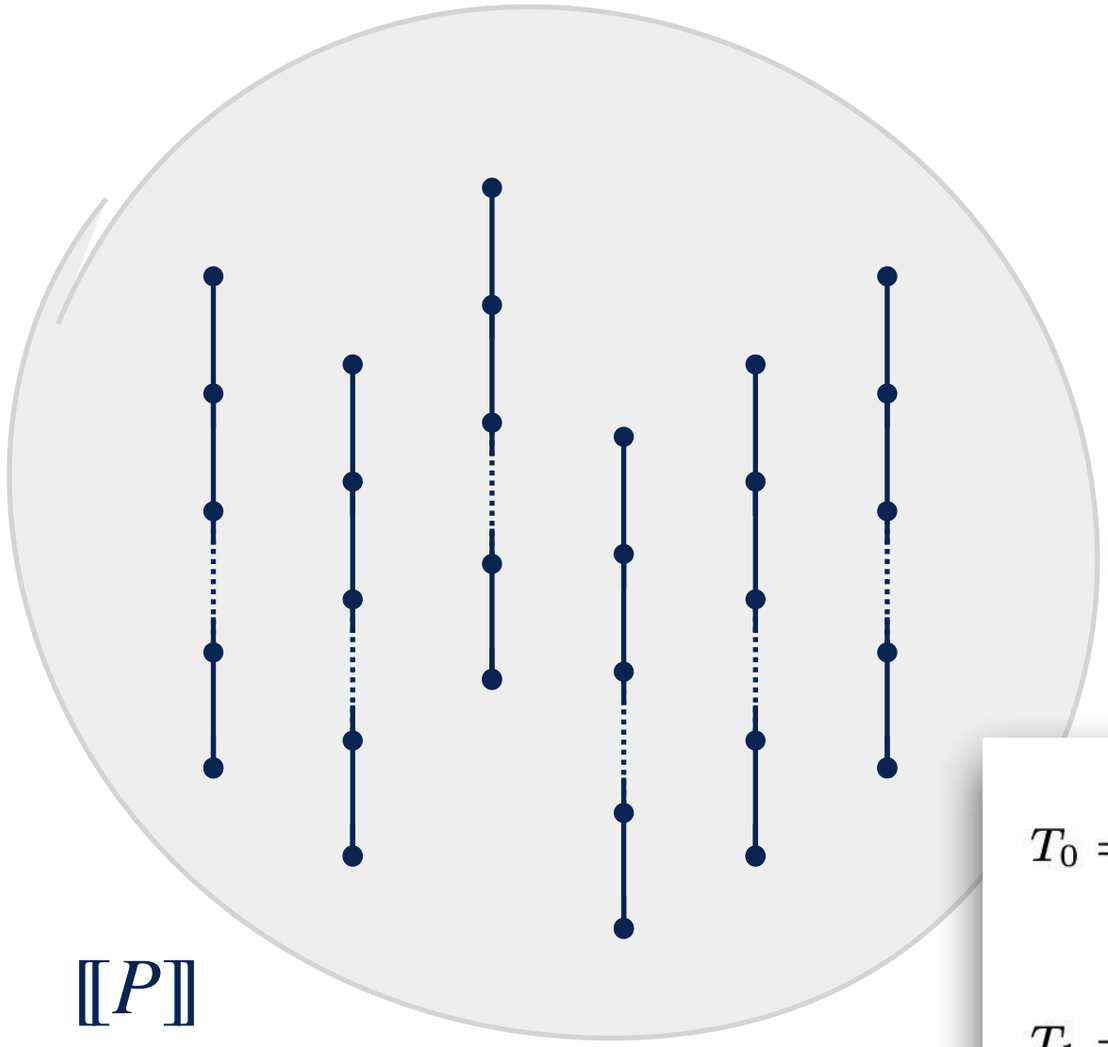
practical tools
targeting specific programs

algorithmic approaches
to decide program properties



mathematical models
of the program behavior

Maximal Trace Semantics



$[\![P]\!]$

Least fixpoint formulation of maximal traces

Idea: Forge a least fixpoint formulation for whole $\mathcal{M}_\infty$, we merge finite and infinite maximal trace least fixpoint forms

Fixpoint fusion:

$\mathcal{M}_\infty \cap \Sigma^*$ is best defined on $(\mathcal{P}(\Sigma^*), \subseteq, \cup, \cap, \emptyset, \Sigma^*)$.
$\mathcal{M}_\infty \cap \Sigma^\omega$ is best defined on $(\mathcal{P}(\Sigma^\omega), \supseteq, \cap, \cup, \Sigma^\omega, \emptyset)$, the dual lattice.
(we transform the greatest fixpoint into a least fixpoint!)

We mix them into a new complete lattice $(\mathcal{P}(\Sigma^\infty), \sqsubseteq, \sqcup, \sqcap, \bot, \top)$:

- $A \sqsubseteq B \overset{\text{def}}{\iff} (A \cap \Sigma^*) \subseteq (B \cap \Sigma^*) \wedge (A \cap \Sigma^\omega) \supseteq (B \cap \Sigma^\omega)$
- $A \sqcup B \overset{\text{def}}{=} ((A \cap \Sigma^*) \cup (B \cap \Sigma^*)) \cup ((A \cap \Sigma^\omega) \cap (B \cap \Sigma^\omega))$
- $A \sqcap B \overset{\text{def}}{=} ((A \cap \Sigma^*) \cap (B \cap \Sigma^*)) \cup ((A \cap \Sigma^\omega) \cup (B \cap \Sigma^\omega))$
- $\bot \overset{\text{def}}{=} \Sigma^\omega$
- $\top \overset{\text{def}}{=} \Sigma^*$

In this lattice, $\mathcal{M}_\infty = \text{lfp}\ F_s$ where $F_s(T) \overset{\text{def}}{=} \mathcal{B} \cup \tau^\frown T$

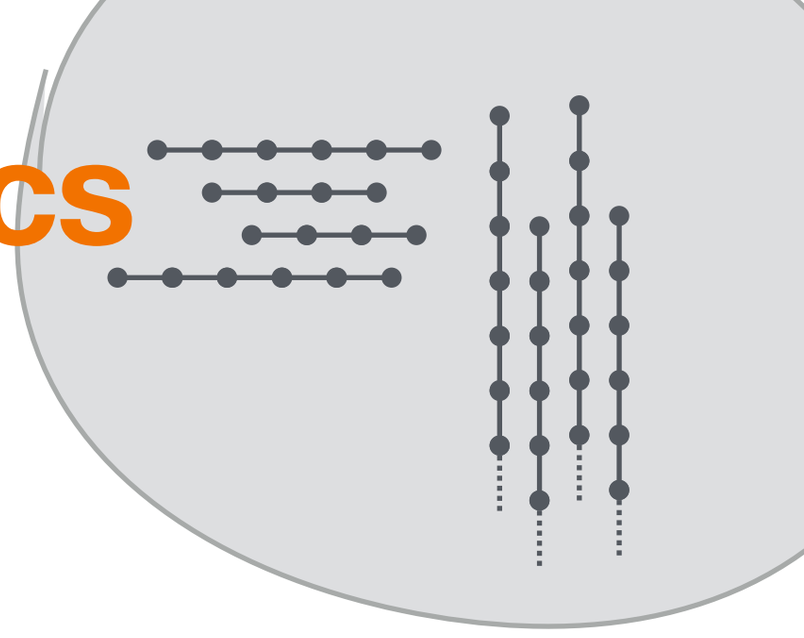
(proof on next slides)

$$T_0 = \left\{ \Sigma^\omega \right\}$$

$$T_1 = \left\{ \Omega \right\} \cup \left\{ \tau \; \Sigma^\omega \right\}$$

$$T_2 = \left\{ \Omega \right\} \cup \left\{ \tau \; \Omega \right\} \cup \left\{ \tau \; \tau \; \Sigma^\omega \right\}$$

Maximal Trace Semantics



$[\![P]\!]$

```
passing = True
if not english:
    english = False
if not math:
    passing = False or bonus
if not math:
    passing = False or bonus
```

passing = **True**

english → _	english → _
math → **T**	math → **T**
science → _	science → _
bonus → _	bonus → _
passing → **?**	passing → **T**

passing = **True** | passing = **False or** bonus | passing = **False or** bonus

english → _	english → _	english → _	english → _
math → **F**	math → **F**	math → **F**	math → **F**
science → _	science → _	science → _	science → _
bonus → **T**	bonus → **T**	bonus → **T**	bonus → **T**
passing → **?**	passing → **T**	passing → **T**	passing → **T**

passing = **True** | passing = **False or** bonus | passing = **False or** bonus

english → _	english → _	english → _	english → _
math → **F**	math → **F**	math → **F**	math → **F**
science → _	science → _	science → _	science → _
bonus → **F**	bonus → **F**	bonus → **F**	bonus → **F**
passing → **?**	passing → **T**	passing → **F**	passing → **F**

Input Data (Non-)Usage

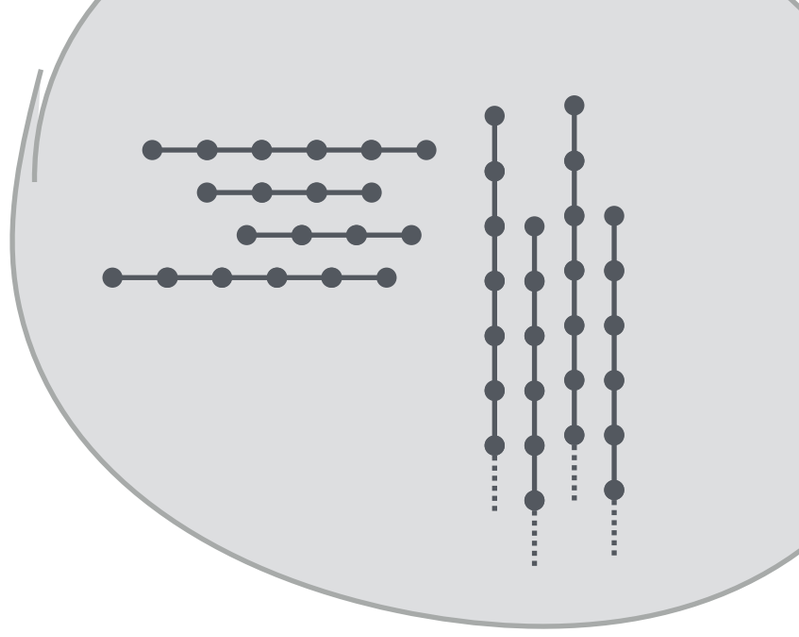
$$\mathcal{N}_J \overset{\text{def}}{=} \{[\![P]\!] \in \mathcal{P}(\Sigma^{+\infty}) \mid \forall i \in J \subseteq I_P : \text{UNUSED}_i([\![P]\!])\}$$

$\mathcal{N}_J$ is the set of all programs P (or, rather, their semantics $[\![P]\!]$) that **do not use** the value of the input variables in $J \subseteq I_P$

$$\begin{aligned}\text{UNUSED}_i([\![M]\!]) \overset{\text{def}}{=} \; & \forall t \in [\![P]\!], v \in \mathcal{V} : t_0(i) \neq v \Rightarrow \exists t' \in [\![P]\!] : \\ & (\forall 0 \leq j \leq |I_P| : j \neq i \Rightarrow t_0(j) = t'_0(j)) \\ & \wedge\, t'_0(i) = v \\ & \wedge\, t_\omega = t'_\omega\end{aligned}$$

Intuitively: **any possible program outcome** is possible **from any value of the input** variable $x_{0,i}$

Input Data (Non-)Usage



$[\![P]\!]$

```
passing = True
if not english:
    english = False
if not math:
    passing = False or bonus
if not math:
    passing = False or bonus
```

passing = **True**

english → _	english → _
math → **T**	math → **T**
science → _	science → _
bonus → _	bonus → _
passing → **?**	passing → **T**

passing = **True** | passing = **False or** bonus | passing = **False or** bonus

english → _	english → _	english → _	english → _
math → **F**	math → **F**	math → **F**	math → **F**
science → _	science → _	science → _	science → _
bonus → **T**	bonus → **T**	bonus → **T**	bonus → **T**
passing → **?**	passing → **T**	passing → **T**	passing → **T**

passing = **True** | passing = **False or** bonus | passing = **False or** bonus

english → _	english → _	english → _	english → _
math → **F**	math → **F**	math → **F**	math → **F**
science → _	science → _	science → _	science → _
bonus → **F**	bonus → **F**	bonus → **F**	bonus → **F**
passing → **?**	passing → **T**	passing → **F**	passing → **F**

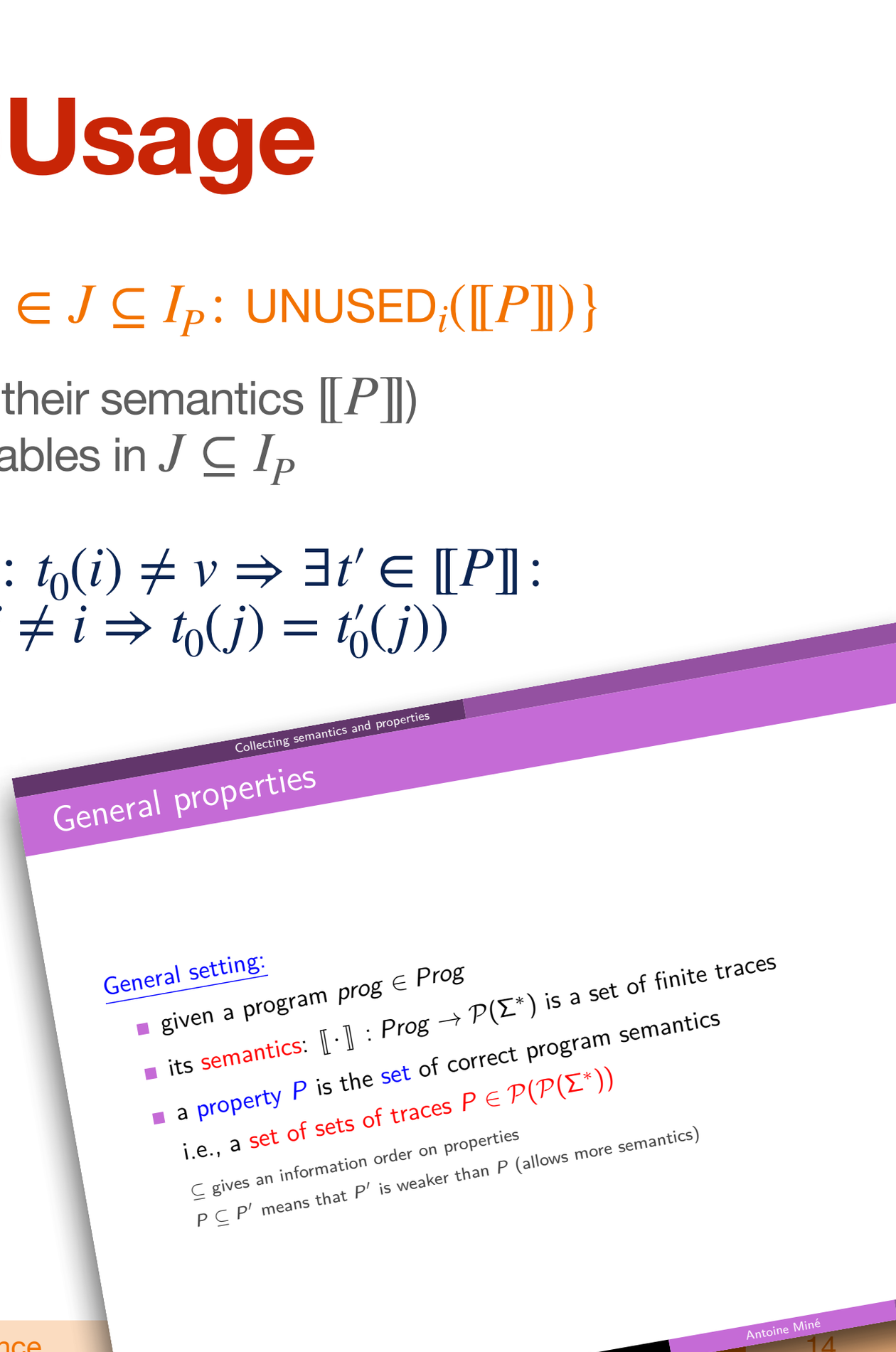
Input Data (Non-)Usage

$$\mathcal{N}_J \stackrel{\text{def}}{=} \{ [\![P]\!] \in \mathcal{P}(\Sigma^{+\infty}) \mid \forall i \in J \subseteq I_P : \text{UNUSED}_i([\![P]\!]) \}$$

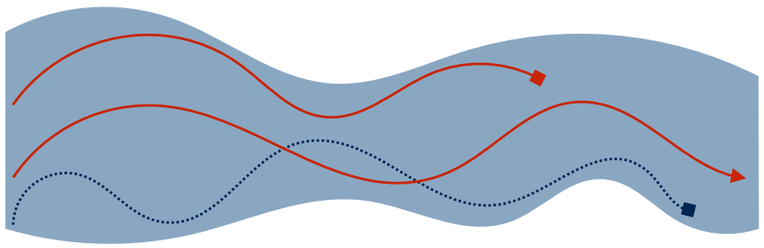
$\mathcal{N}_J$ is the set of all programs P (or, rather, their semantics $[\![P]\!]$) that **do not use** the value of the input variables in $J \subseteq I_P$

$$\begin{aligned}\text{UNUSED}_i([\![M]\!]) \stackrel{\text{def}}{=} \; & \forall t \in [\![P]\!], v \in \mathcal{V} : t_0(i) \neq v \Rightarrow \exists t' \in [\![P]\!] : \\ & (\forall 0 \leq j \leq |I_P| : j \neq i \Rightarrow t_0(j) = t'_0(j)) \\ & \wedge\, t'_0(i) = v \\ & \wedge\, t_\omega = t'_\omega\end{aligned}$$

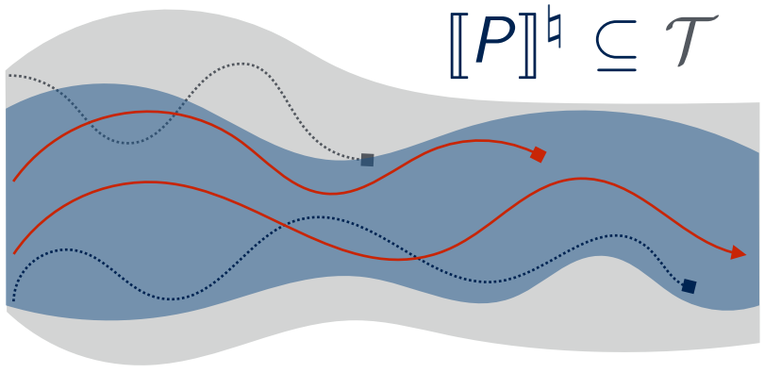
Intuitively: **any possible program outcome** is possible **from any value of the input** variable $x_{0,i}$



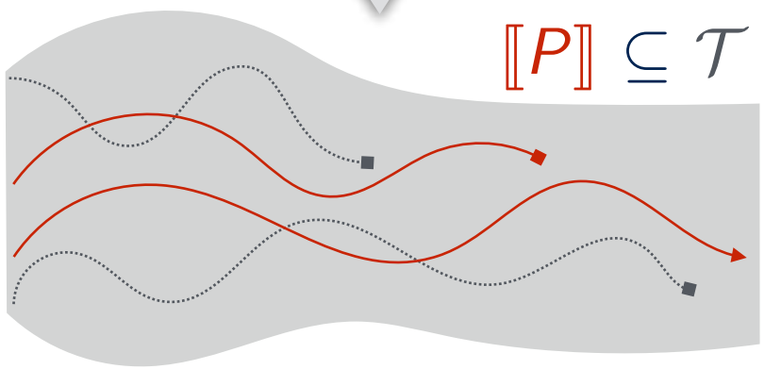
Trace Properties



$[\![P]\!] \subseteq [\![P]\!]^{\natural}$



$[\![P]\!]^{\natural} \subseteq \mathcal{T}$



$[\![P]\!] \subseteq \mathcal{T}$

Collecting semantics and properties

Restricted properties

Reasoning on (and abstracting) $\mathcal{P}(\mathcal{P}(\Sigma^*))$ is hard!

In the following, we use a simpler setting:

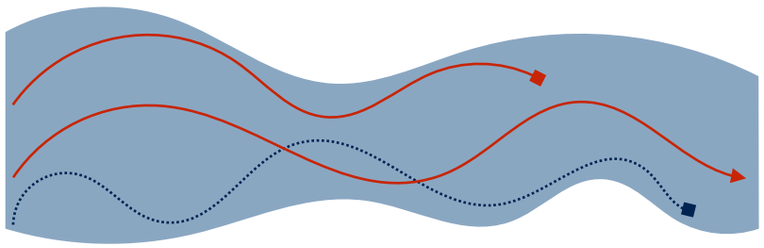
- a property is a set of traces $P \in \mathcal{P}(\Sigma^*)$
- the collecting semantics is a set of traces: $Col(prog) \stackrel{\text{def}}{=} [\![prog]\!]$
- the verification problem remains an inclusion check: $[\![prog]\!] \subseteq P$
- abstractions will over-approximate the set of traces $[\![prog]\!]$

Example properties:

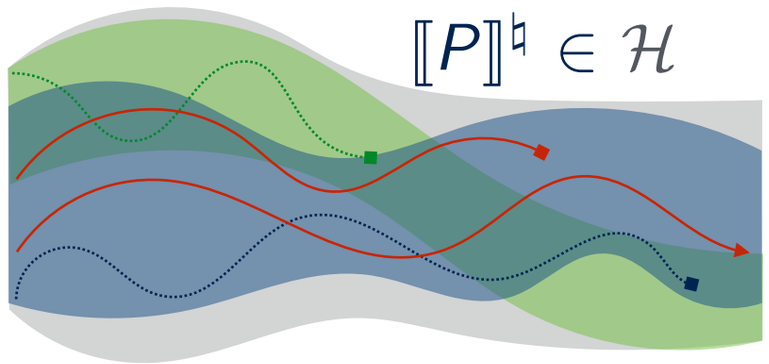
- state property $P \stackrel{\text{def}}{=} S^*$ (remains in the set S of safe states)
- maximal execution time: $P \stackrel{\text{def}}{=} S^{\leq k}$
- ordering: $P \stackrel{\text{def}}{=} (\Sigma \setminus \{b\})^* \cdot a \cdot \Sigma^* \cdot b \cdot \Sigma^*$ (a occurs before b)

Course 2 | Program Semantics and Properties | Antoine Miné | p. 25 / 98

Program Properties

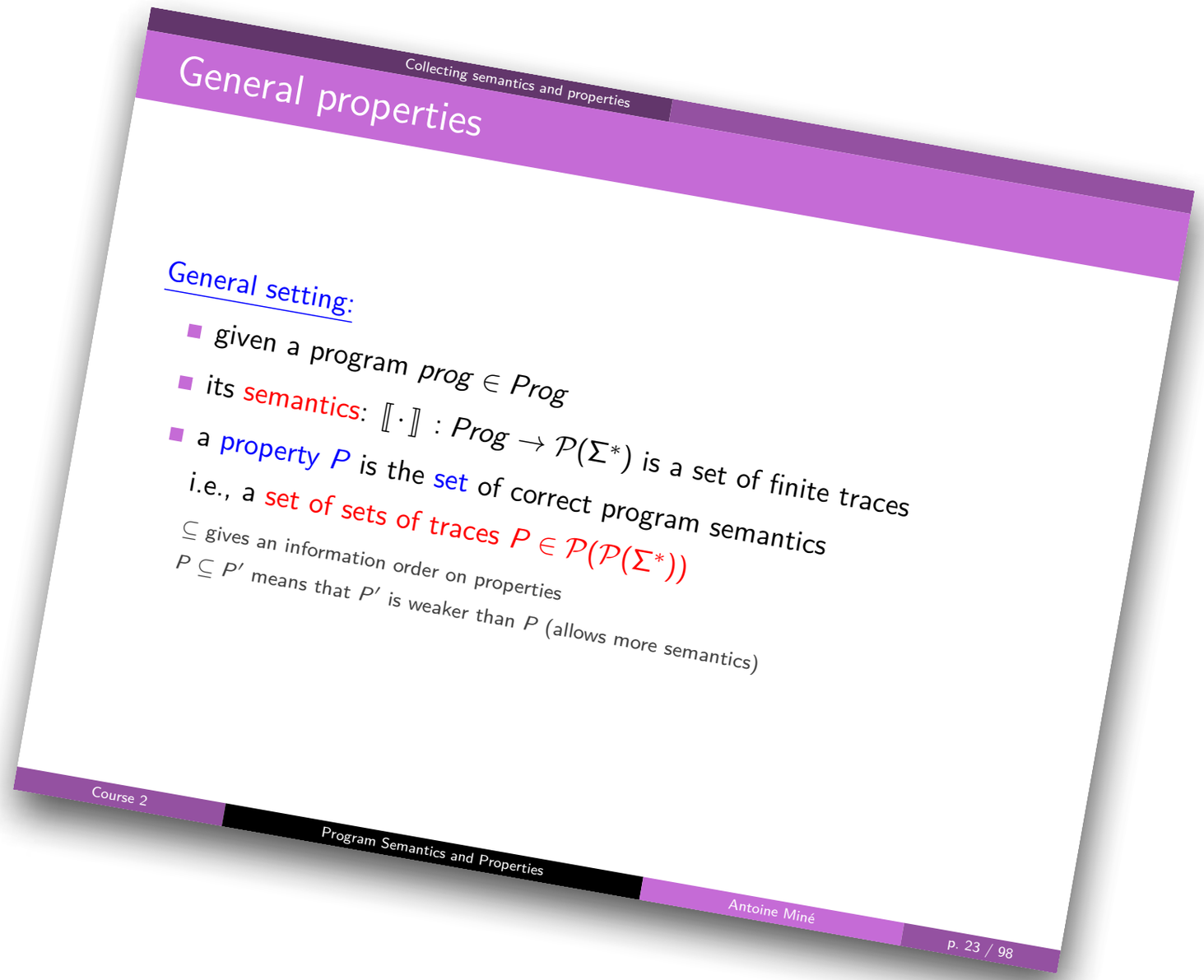


$[\![P]\!] \subseteq [\![P]\!]^\natural$

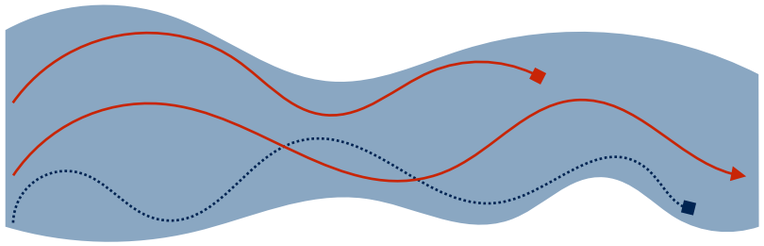


$[\![P]\!]^\natural \in \mathcal{H}$

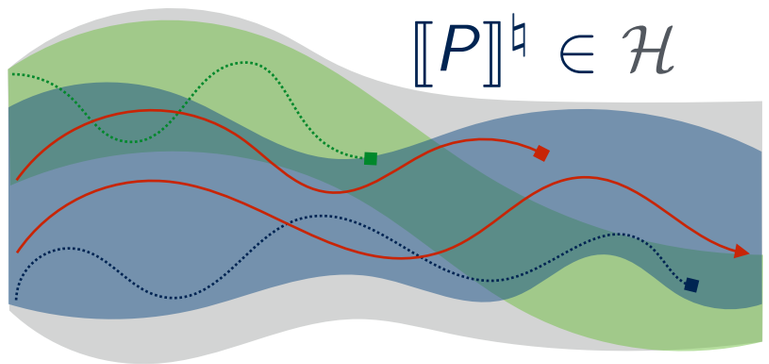
$[\![P]\!] \in \mathcal{H}$

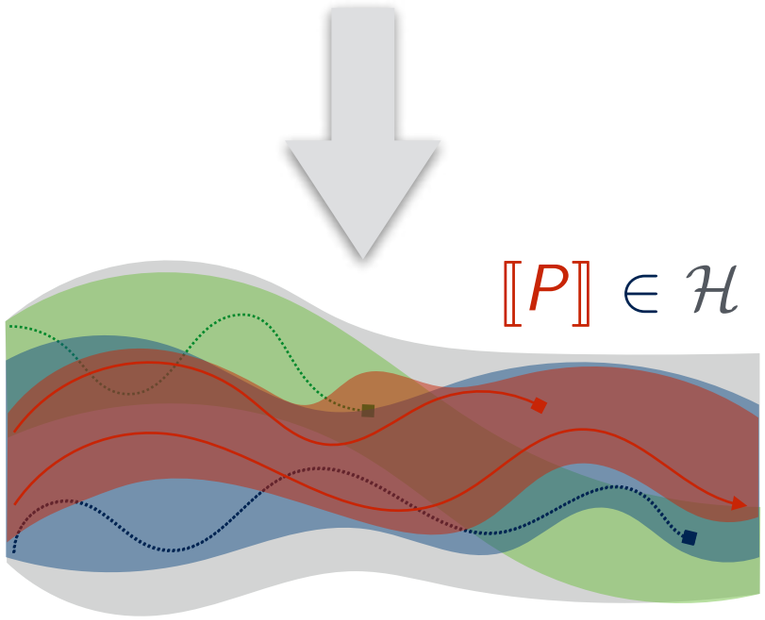


Subset-Closed Properties



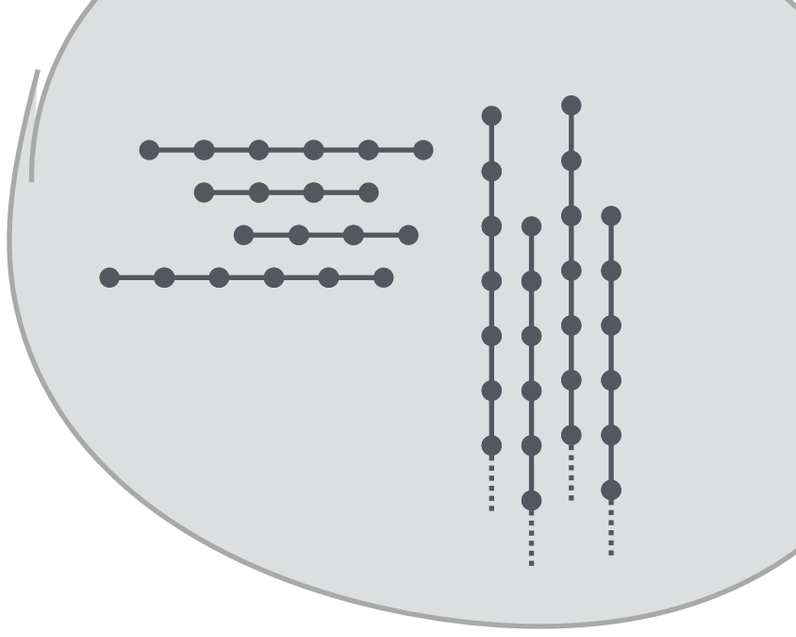
$[\![P]\!] \subseteq [\![P]\!]^\natural$





Input Data (Non-)Usage

```
passing = True
if not english:
    english = False
if not math:
    passing = False or bonus
if not math:
    passing = False or bonus
```



$[\![P]\!]$

passing = **True**

english → _	english → _
math → **T**	math → **T**
science → _	science → _
bonus → _	bonus → _
passing → **?**	passing → **T**

passing = **True** | passing = **False or** bonus | passing = **False or** bonus

english → _	english → _	english → _	english → _
math → **F**	math → **F**	math → **F**	math → **F**
science → _	science → _	science → _	science → _
bonus → **T**	bonus → **T**	bonus → **T**	bonus → **T**
passing → **?**	passing → **T**	passing → **T**	passing → **T**

passing = **True** | passing = **False or** bonus | passing = **False or** bonus

english → _	english → _	english → _	english → _
math → **F**	math → **F**	math → **F**	math → **F**
science → _	science → _	science → _	science → _
bonus → **F**	bonus → **F**	bonus → **F**	bonus → **F**
passing → **?**	passing → **T**	passing → **F**	passing → **F**

Input Data (Non-)Usage

$$\mathcal{N}_J \stackrel{\text{def}}{=} \{ [\![P]\!] \in \mathcal{P}(\Sigma^{+\infty}) \mid \text{UNUSED}_J([\![P]\!]) \}$$

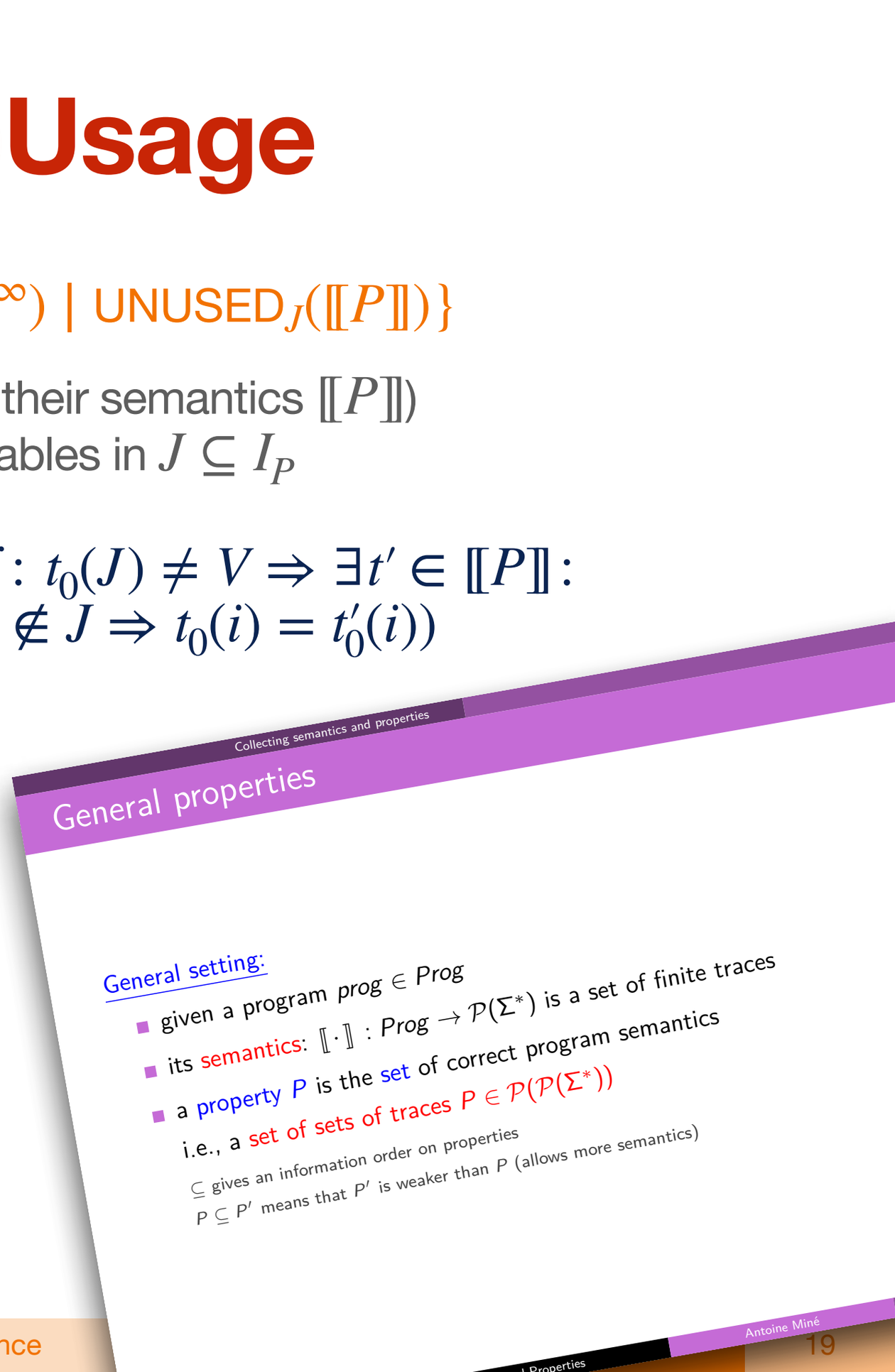
$\mathcal{N}_J$ is the set of all programs P (or, rather, their semantics $[\![P]\!]$) that **do not use** the value of the input variables in $J \subseteq I_P$

$$\begin{aligned}\text{UNUSED}_J([\![M]\!]) \stackrel{\text{def}}{=} & \forall t \in [\![P]\!], V \in \mathcal{V} : t_0(J) \neq V \Rightarrow \exists t' \in [\![P]\!] : \\ & (\forall 0 \leq i \leq |I_P| : i \notin J \Rightarrow t_0(i) = t'_0(i)) \\ & \wedge\, t'_0(J) = V \\ & \wedge\, t_\omega = t'_\omega\end{aligned}$$

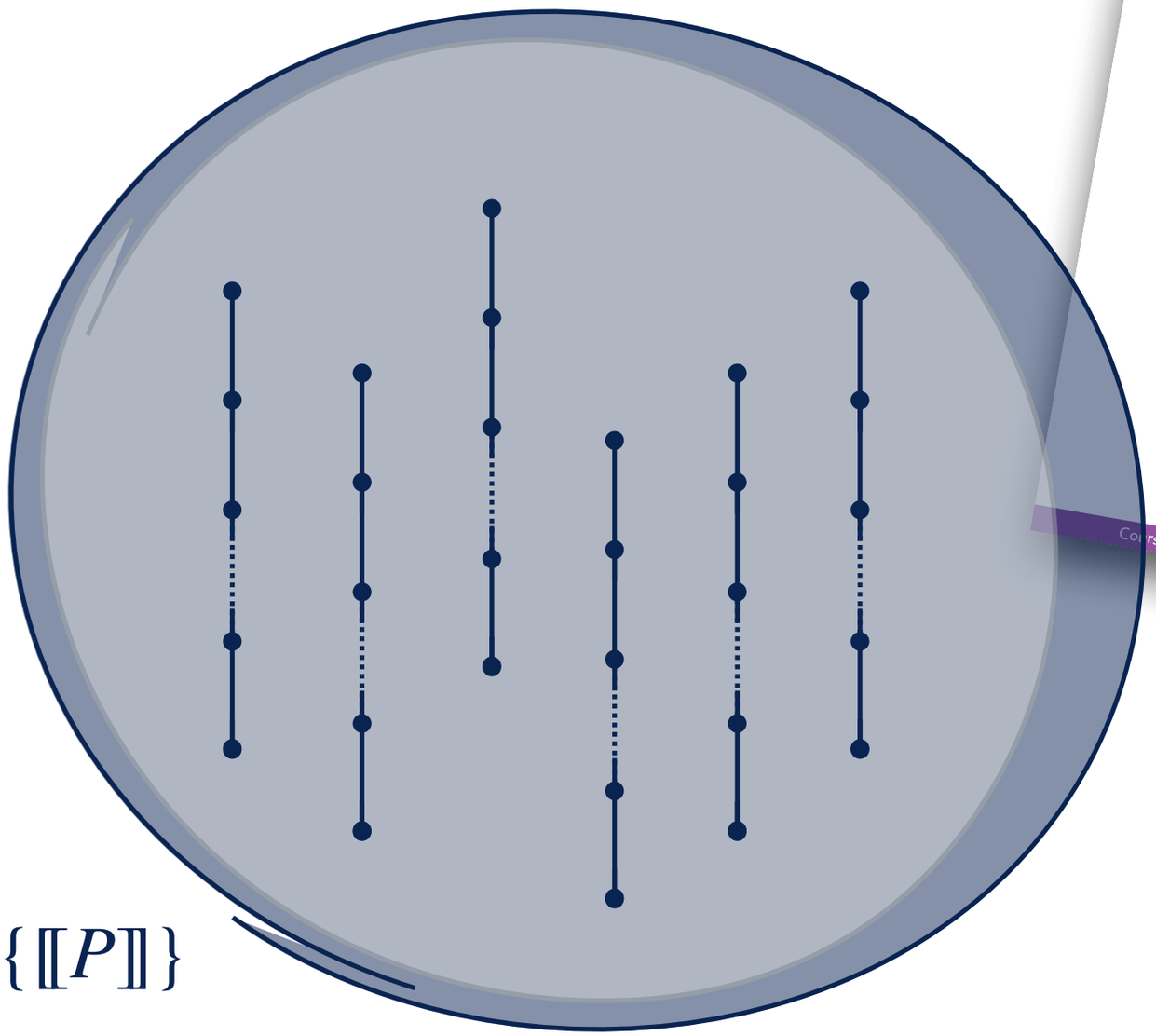
Intuitively: **any possible program outcome** is possible **from any value of the input** variables in J

Theorem

$$P \models \mathcal{N}_J \Leftrightarrow \{ [\![P]\!] \} \subseteq \mathcal{N}_J$$



Collecting Semantics



General collecting semantics

The collecting semantics $Col : Prog \to \mathcal{P}(\mathcal{P}(\Sigma^*))$
is the strongest property of a program

Hence: $Col(prog) \stackrel{\text{def}}{=} \{[\![prog]\!]\}$

Benefits: uniformity of semantics and properties, $\subseteq$ information order

- given a program *prog* and a property $P \in \mathcal{P}(\mathcal{P}(\Sigma^*))$
  the verification problem is an inclusion check:

  $$Col(prog) \subseteq P$$

- generally, the collecting semantics cannot be computed,
  we settle for a weaker property $S^\sharp$ that
  - is sound: $Col(prog) \subseteq S^\sharp$
  - implies the desired property: $S^\sharp \subseteq P$

Collecting Semantics

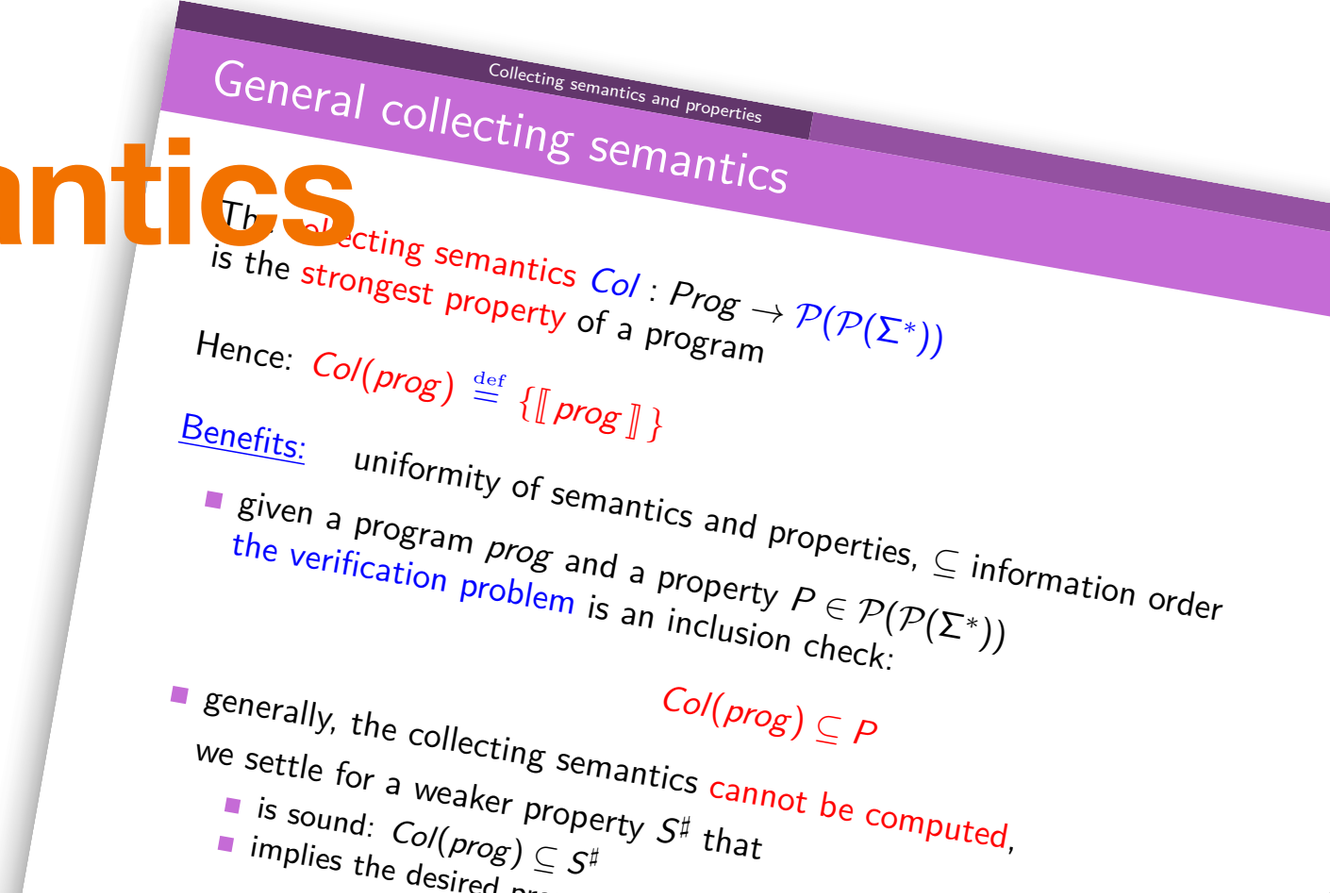
Intuition

Property (*by extension*): set of elements that have that property

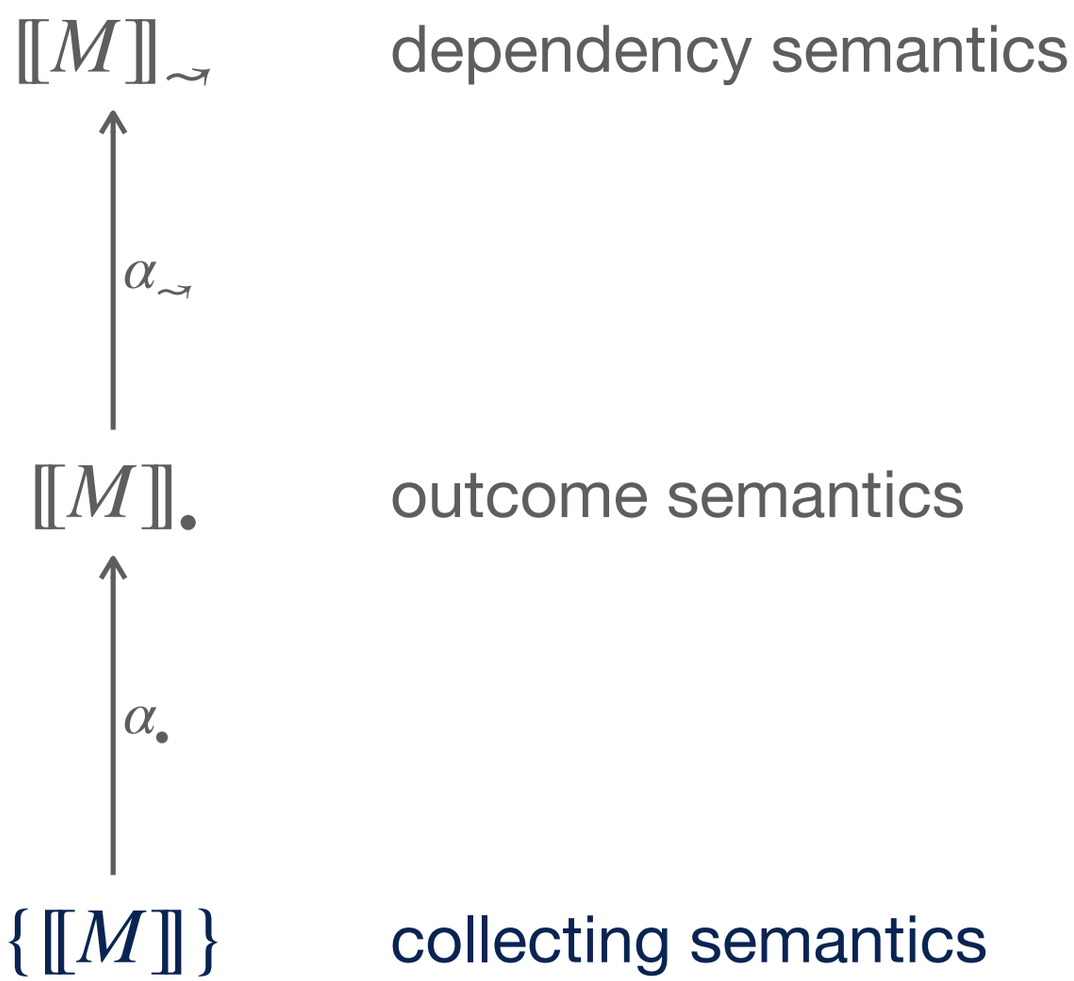
Property “being Patrick Cousot”

Property “being program P”

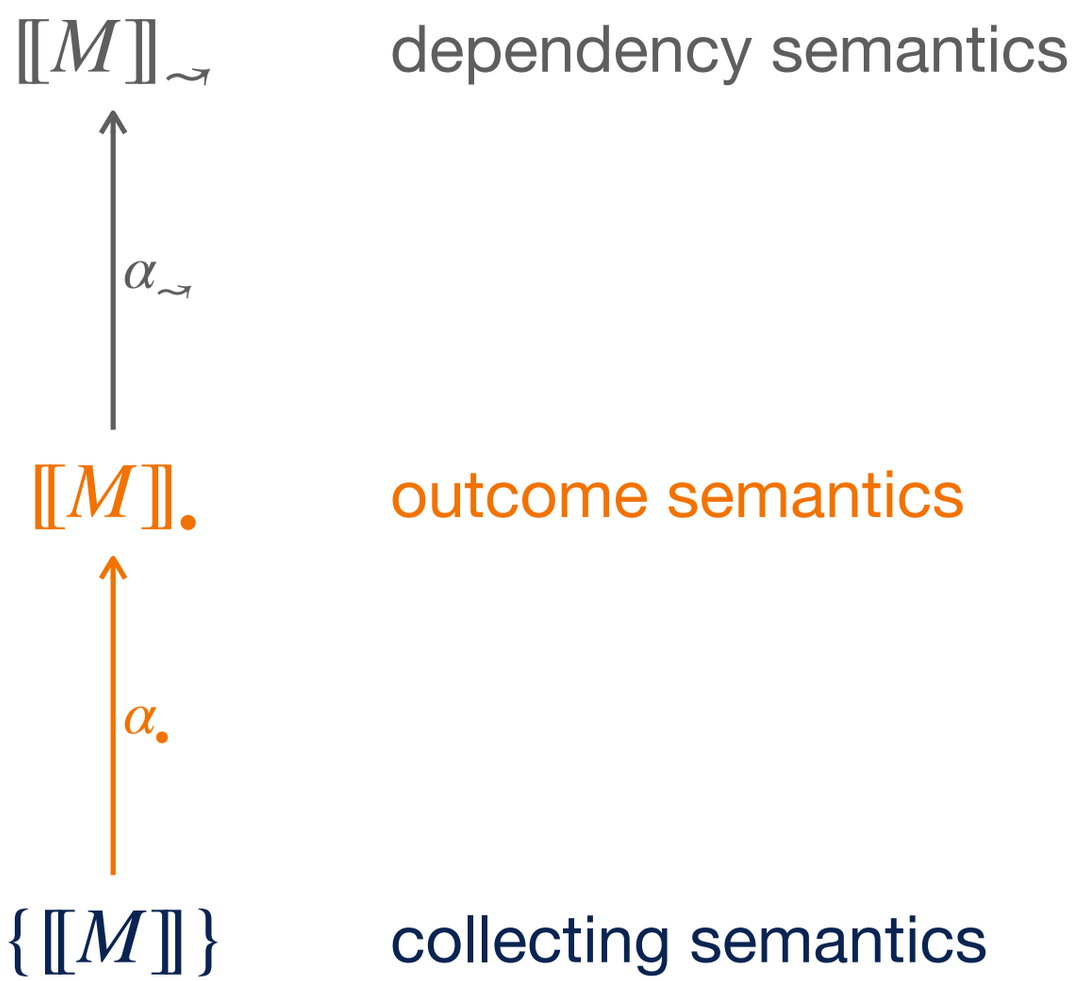
$$\{[\![P]\!]\}$$



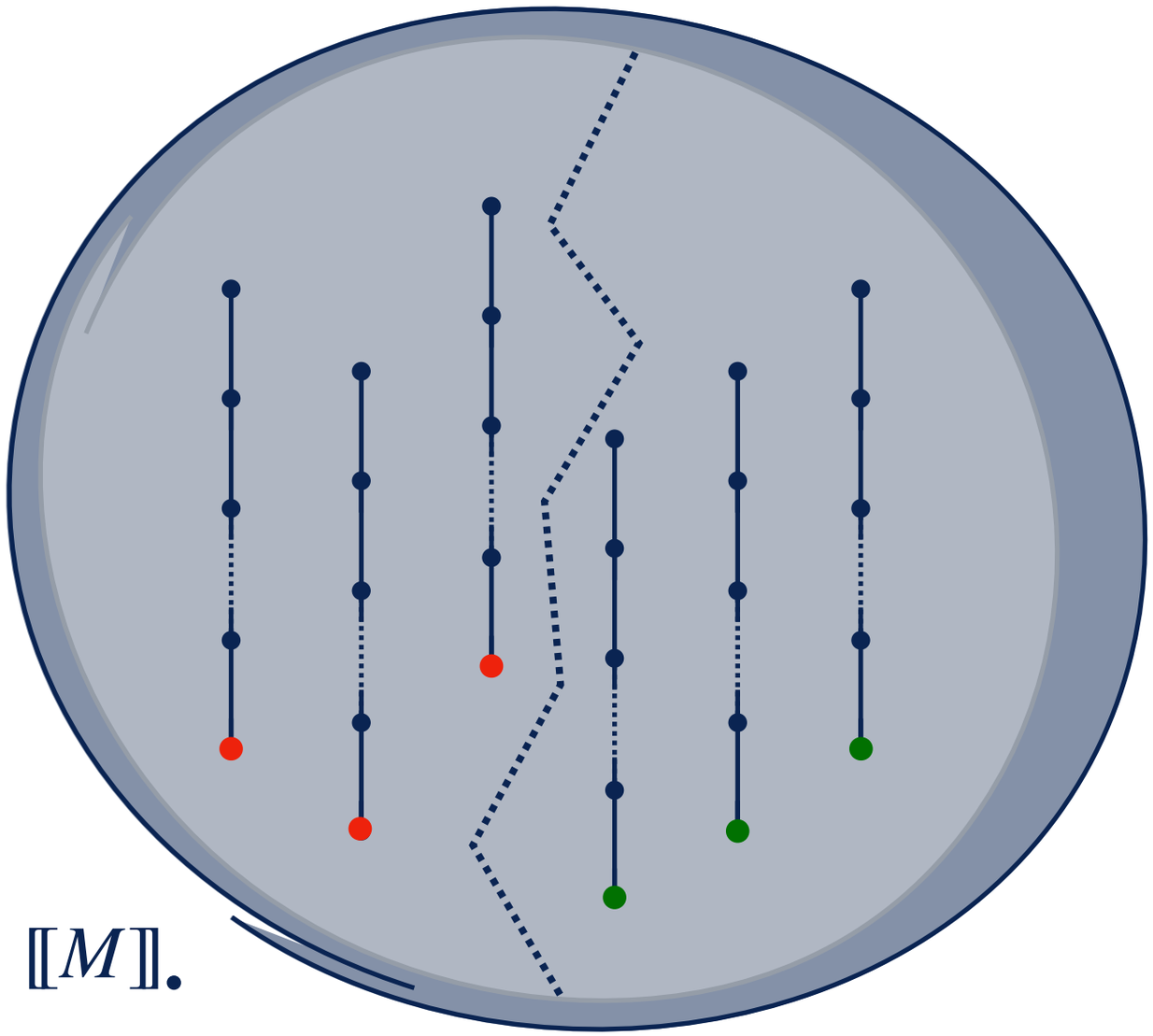
(Another) Hierarchy of Semantics



(Another) Hierarchy of Semantics



Outcome Semantics



$[\![M]\!]$.

partitioning a set of traces that satisfies input data (non-)usage **with respect to the program outcome** yields sets of traces that also satisfy input data (non-)usage

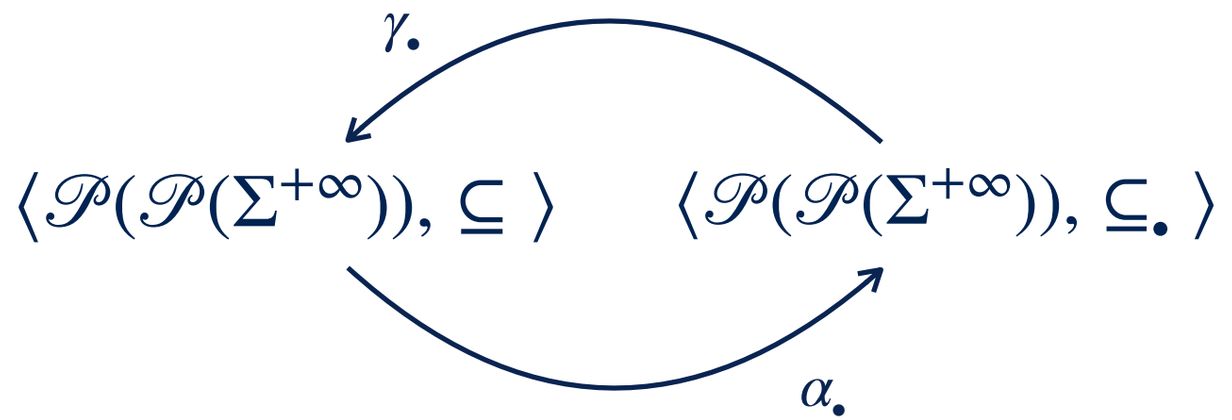
Outcome Semantics

$$\mathbb{O} \stackrel{\text{def}}{=} \{\Sigma^{+}_{o_1=v_1,\ldots,o_k=v_k} \mid v_1, \ldots, v_k \in \mathscr{V}\} \cup \{\Sigma^{\omega}\}$$

outcomes

Lemma

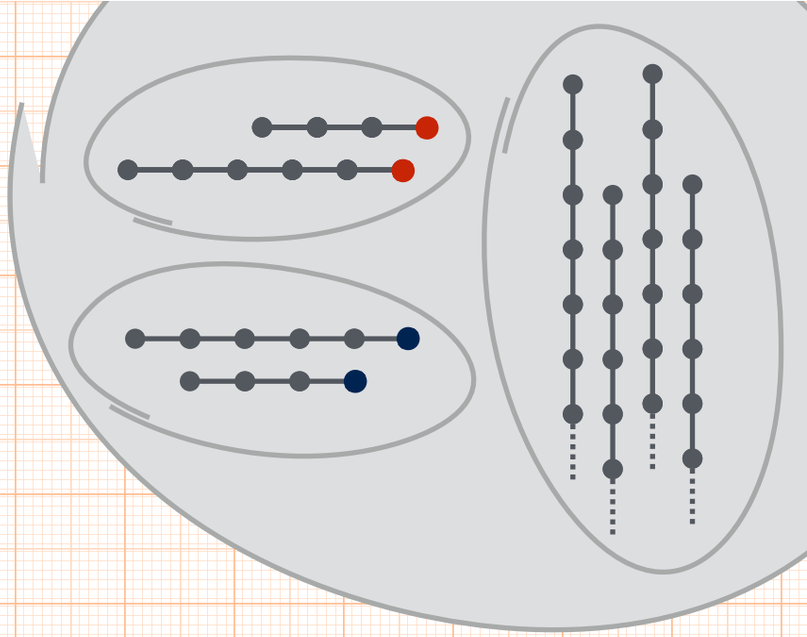
$$P \vDash \mathscr{N}_J \Leftrightarrow \{[\![P]\!] \cap O \mid O \in \mathbb{O}\} \subseteq \mathscr{N}_J$$



$$\alpha_\bullet(S) \stackrel{\text{def}}{=} \{T \cap O \mid T \in S \wedge O \in \mathbb{O}\}$$

outcome abstraction

Outcome Semantics



$[\![P]\!]_\bullet$

```
passing = True
if not english:
    english = False
if not math:
    passing = False or bonus
if not math:
    passing = False or bonus
```

passing = **True**

english → _	english → _
math → **T**	math → **T**
science → _	science → _
bonus → _	bonus → _
passing → **?**	passing → **T**

passing = **True** | passing = **False or** bonus | passing = **False or** bonus

english → _	english → _	english → _	english → _
math → **F**	math → **F**	math → **F**	math → **F**
science → _	science → _	science → _	science → _
bonus → **T**	bonus → **T**	bonus → **T**	bonus → **T**
passing → **?**	passing → **T**	passing → **T**	passing → **T**

passing = **True** | passing = **False or** bonus | passing = **False or** bonus

english → _	english → _	english → _	english → _
math → **F**	math → **F**	math → **F**	math → **F**
science → _	science → _	science → _	science → _
bonus → **F**	bonus → **F**	bonus → **F**	bonus → **F**
passing → **?**	passing → **T**	passing → **F**	passing → **F**

$\emptyset$

unfeasible non-termination outcome

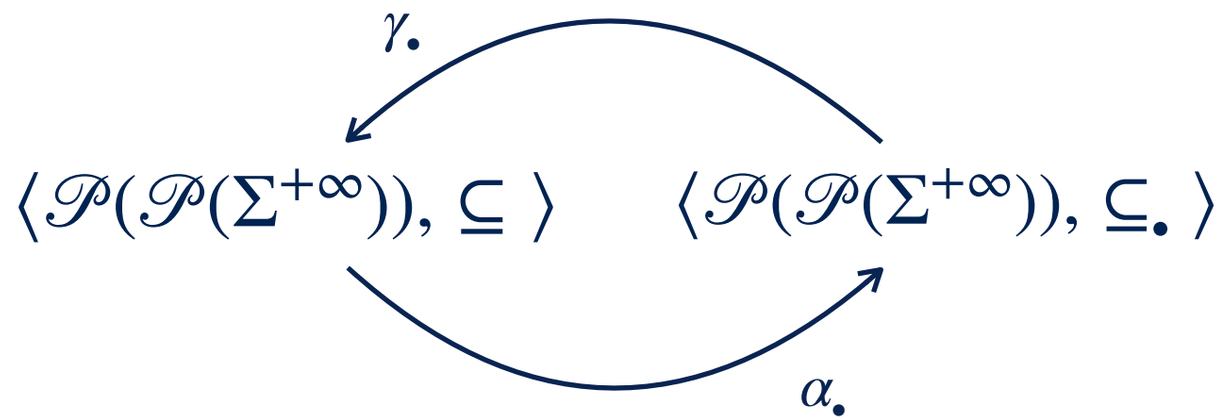
Outcome Semantics

$$\mathbb{O} \stackrel{\text{def}}{=} \{\Sigma^{+}_{o_1=v_1,\ldots,o_k=v_k} \mid v_1, \ldots, v_k \in \mathscr{V}\} \cup \{\Sigma^{\omega}\}$$

outcomes

Lemma

$$P \vDash \mathscr{N}_J \Leftrightarrow \{[\![P]\!] \cap O \mid O \in \mathbb{O}\} \subseteq \mathscr{N}_J$$

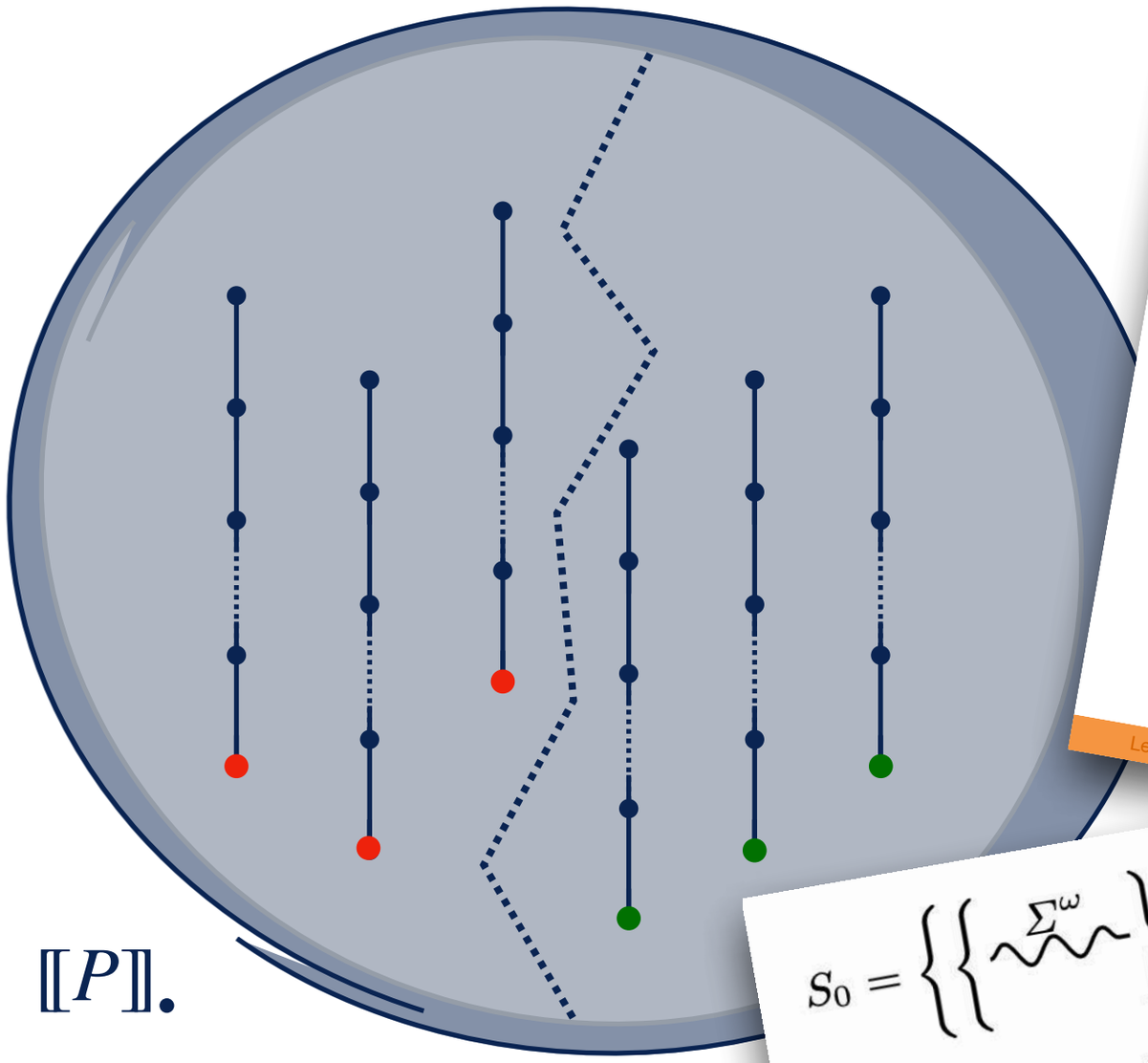


$$\alpha_\bullet(S) \stackrel{\text{def}}{=} \{T \cap O \mid T \in S \wedge O \in \mathbb{O}\}$$

outcome abstraction

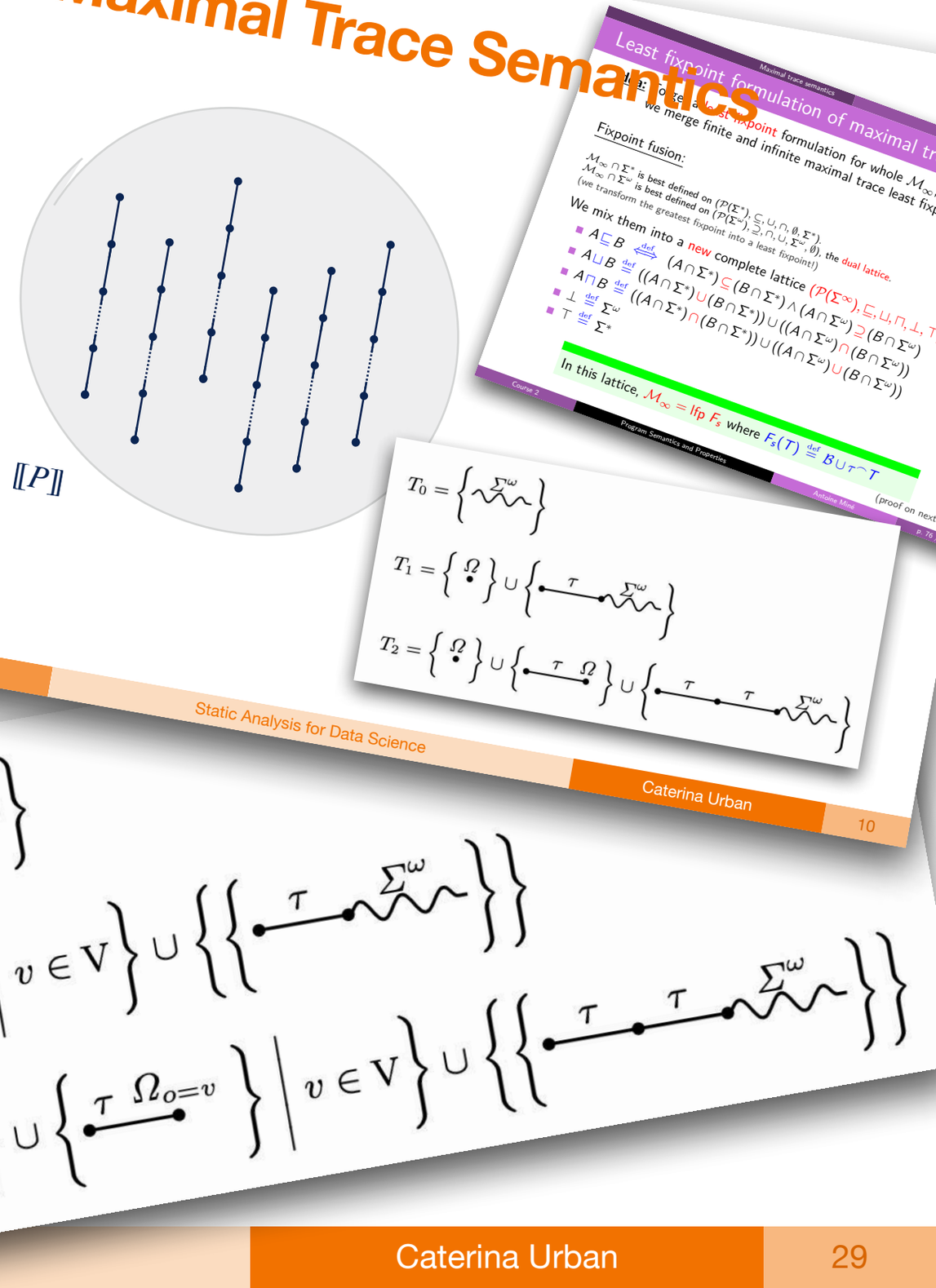
$$[\![P]\!]_\bullet \stackrel{\text{def}}{=} \alpha_\bullet(\{[\![P]\!]\}) = \{[\![P]\!] \cap O \mid O \in \mathbb{O}\}$$

Outcome Semantics



$[\![P]\!]$.

Maximal Trace Semantics



Outcome Semantics

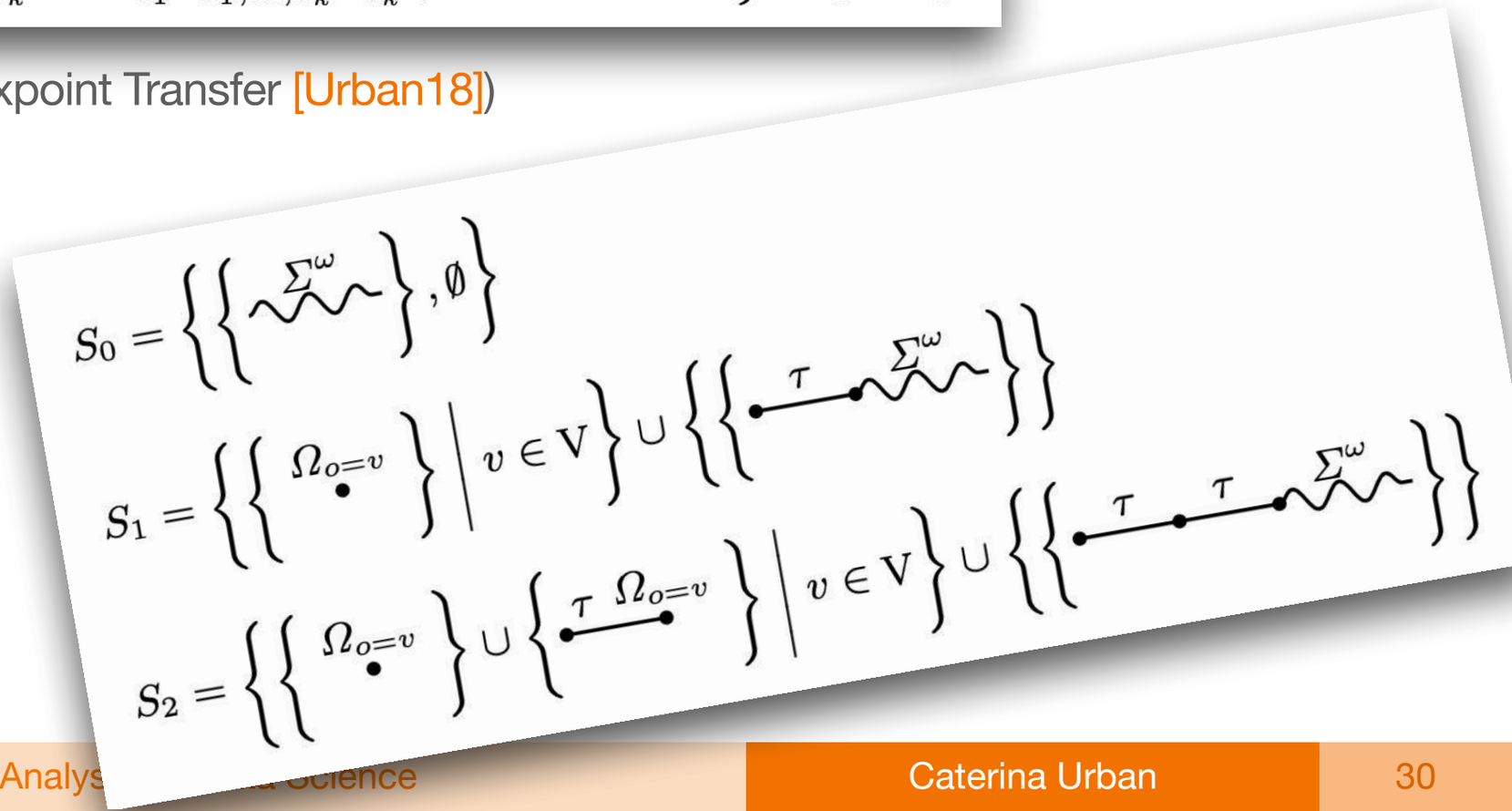
$$S_1 \sqsubseteq S_2 \stackrel{\text{def}}{=} \bigwedge_{v_1,\ldots,v_k \in \mathrm{V}} {S_1}^{+}_{o_1=v_1,\ldots,o_k=v_k} \subseteq {S_2}^{+}_{o_1=v_1,\ldots,o_k=v_k} \wedge S_1^{\omega} \supseteq S_2^{\omega}$$

Theorem 1. *The outcome semantics $\Lambda_\bullet \in \mathcal{P}\left(\mathcal{P}\left(\Sigma^{+\infty}\right)\right)$ can be expressed as a least fixpoint in $\langle \mathcal{P}\left(\mathcal{P}\left(\Sigma^{+\infty}\right)\right), \sqsubseteq, \sqcup, \sqcap, \{\Sigma^\omega, \emptyset\}, \{\emptyset, \Sigma^+\}\rangle$ as:*

$$\begin{aligned} \Lambda_\bullet &= \mathrm{lfp}^{\sqsubseteq} \Theta_\bullet \\ \Theta_\bullet(S) &\stackrel{def}{=} \{\Omega_{o_1=v_1,\ldots,o_k=v_k} \mid v_1,\ldots,v_k \in \mathrm{V}\} \uplus \{\tau\,;T \mid T \in S\} \end{aligned} \tag{9}$$

where $S_1 \uplus S_2 \stackrel{def}{=} \{{S_1}^{+}_{o_1=v_1,\ldots,o_k=v_k} \cup {S_2}^{+}_{o_1=v_1,\ldots,o_k=v_k} \mid v_1,\ldots,v_k \in \mathrm{V}\} \cup S_1^{\omega} \cup S_2^{\omega}$.

(proof by Kleenian Fixpoint Transfer [Urban18])



Input Data (Non-)Usage

$$\mathcal{N}_J \stackrel{\text{def}}{=} \{[\![P]\!] \in \mathcal{P}(\Sigma^{+\infty}) \mid \text{UNUSED}_J([\![P]\!])\}$$

$\mathcal{N}_J$ is the set of all programs P (or, rather, their semantics $[\![P]\!]$) that **do not use** the value of the input variables in $J \subseteq I_P$

$$\text{UNUSED}_J([\![M]\!]) \stackrel{\text{def}}{=} \forall t \in [\![P]\!], V \in \mathcal{V} : t_0(J) \neq V \Rightarrow \exists t' \in [\![P]\!] : (\forall 0 \leq i \leq |I_P| : i \notin J \Rightarrow t_0(i) = t'_0(i))$$

Outcome Semantics

$$\mathbb{O} \stackrel{\text{def}}{=} \{\Sigma^+_{o_1=v_1,\ldots,o_k=v_k} \mid v_1, \ldots, v_k \in \mathcal{V}\} \cup \{\Sigma^\omega\}$$ **outcomes**

Lemma

$$P \vDash \mathcal{N}_J \Leftrightarrow \{[\![P]\!] \cap O \mid O \in \mathbb{O}\} \subseteq \mathcal{N}_J$$

$$\langle \mathcal{P}(\mathcal{P}(\Sigma^{+\infty})), \subseteq \rangle \quad \langle \mathcal{P}(\mathcal{P}(\Sigma^{+\infty})), \subseteq_\bullet \rangle$$
$$\gamma_\bullet \qquad \alpha_\bullet$$

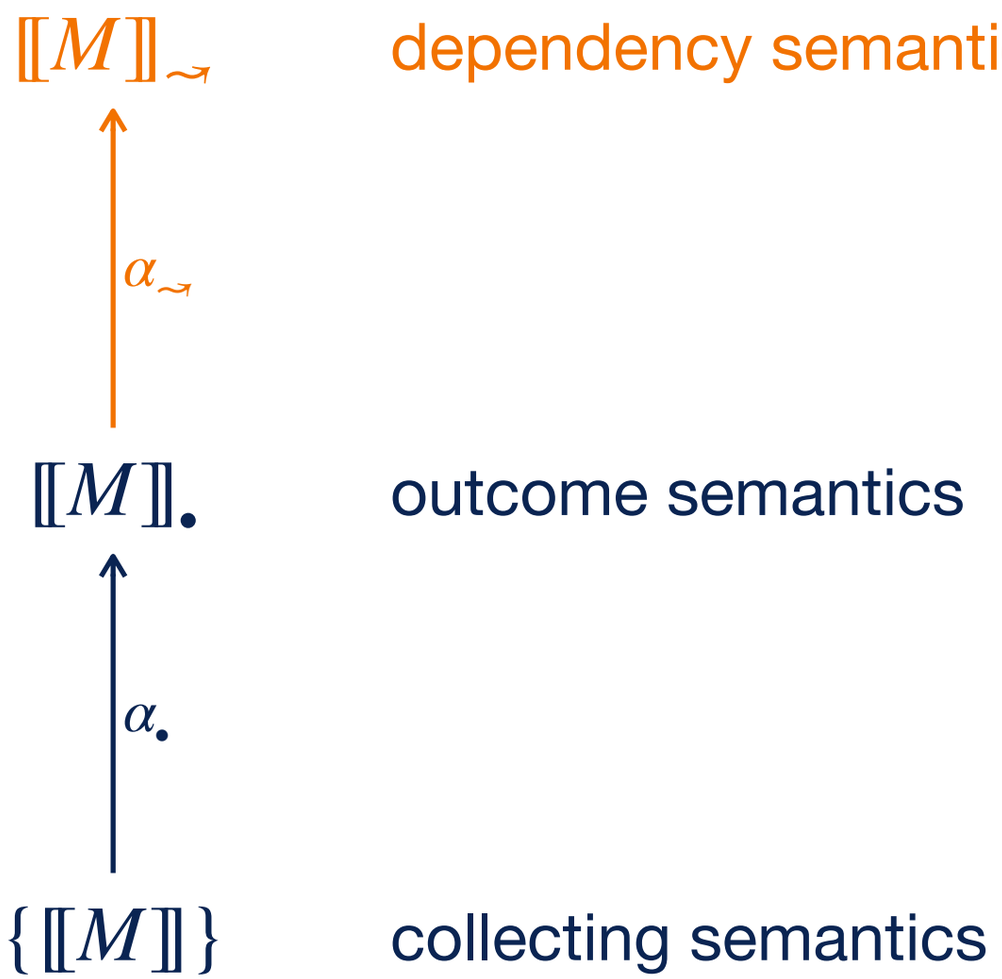
$$\alpha_\bullet(S) \stackrel{\text{def}}{=} \{T \cap O \mid T \in S \wedge O \in \mathbb{O}\}$$ **outcome abstraction**

$$[\![P]\!]_\bullet \stackrel{\text{def}}{=} \alpha_\bullet(\{[\![P]\!]\}) = \{[\![P]\!] \cap O \mid O \in \mathbb{O}\}$$

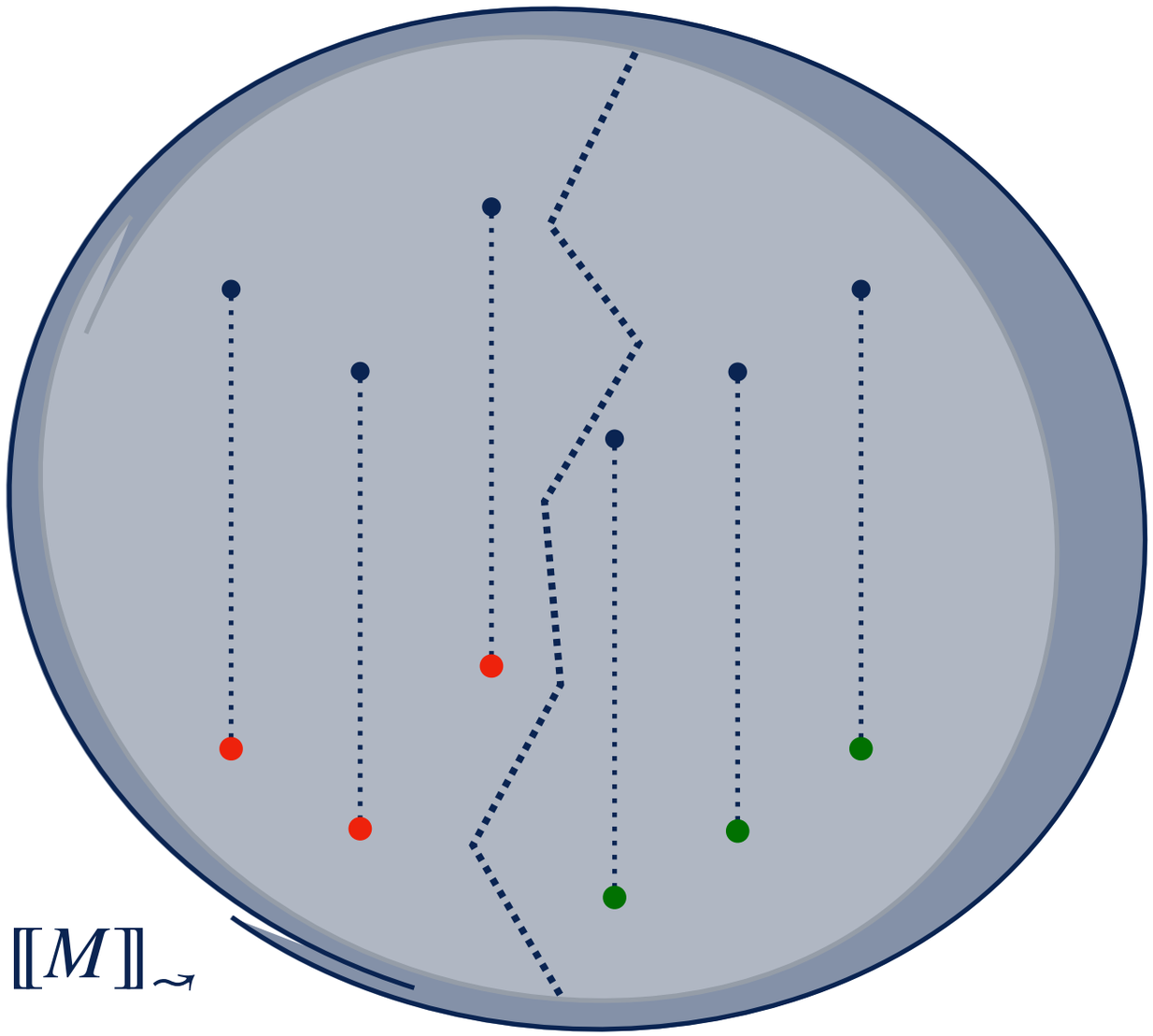
Theorem

$$P \vDash \mathcal{N}_J \Leftrightarrow \{[\![P]\!]\} \subseteq \mathcal{N}_J \Leftrightarrow \alpha_\bullet(\{[\![P]\!]\}) \subseteq \mathcal{N}_J \Leftrightarrow [\![P]\!]_\bullet \subseteq \mathcal{N}_J$$

(Another) Hierarchy of Semantics

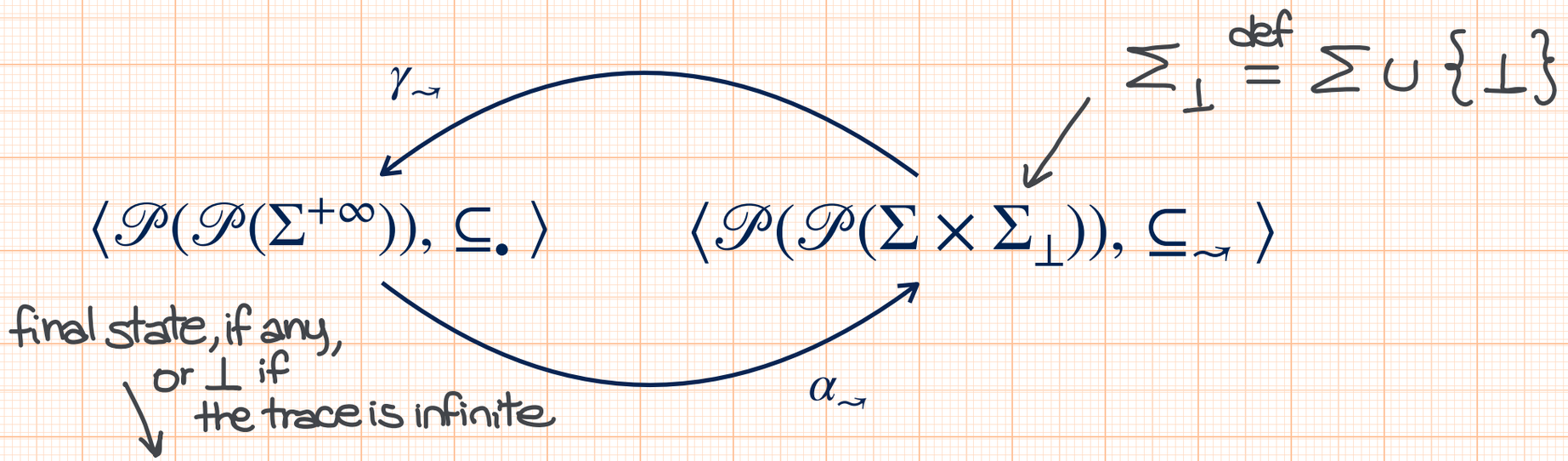


Dependency Semantics



to reason about input data (non-)usage **we do not need to consider all intermediate computations** between the initial and final states of a trace (if any)

Dependency Semantics

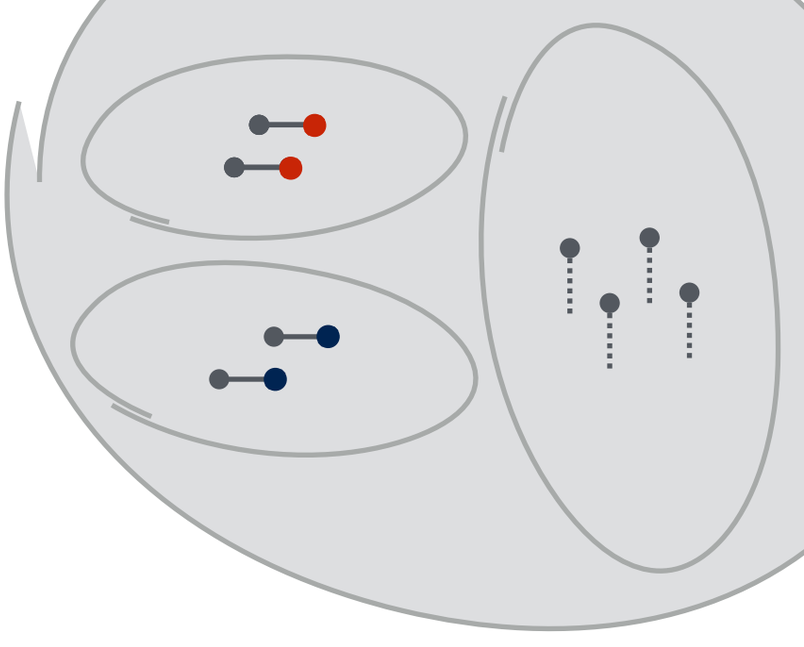


$$\alpha_{\rightsquigarrow}(S) \stackrel{\text{def}}{=} \{\{\langle t_0, t_\omega \rangle \in \Sigma \times \Sigma_\perp \mid t \in T\} \mid T \in S\}$$

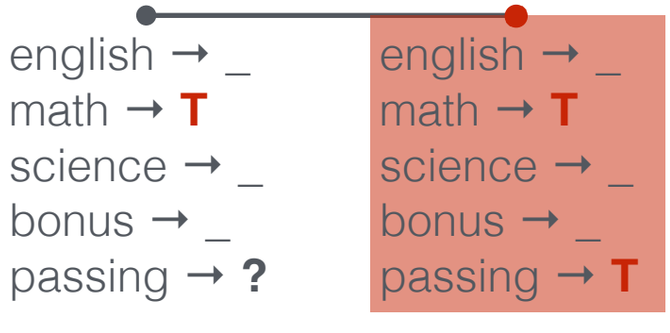
$$\gamma_{\rightsquigarrow}(S) \stackrel{\text{def}}{=} \{T \in \mathscr{P}(\Sigma^{+\infty}) \mid \{\langle t_0, t_\omega \rangle \in \Sigma \times \Sigma_\perp \mid t \in T\} \in S\}$$

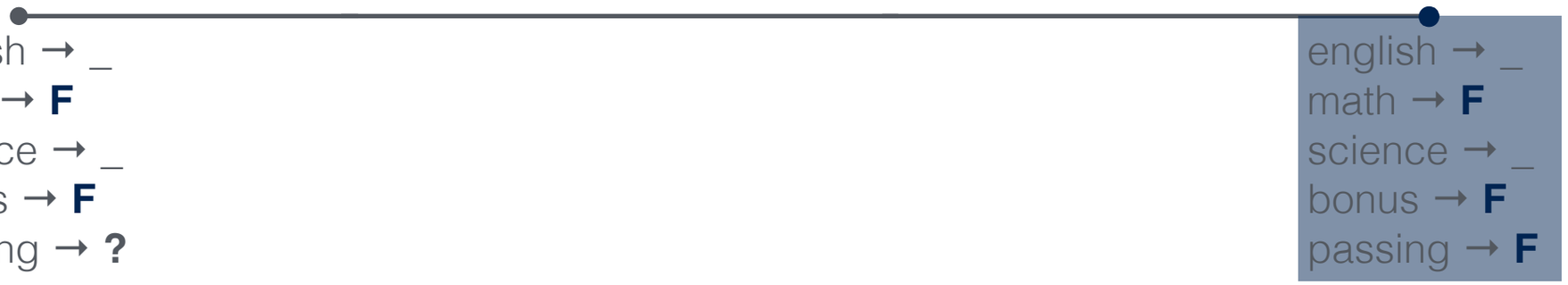
Dependency Semantics

```
passing = True
if not english:
    english = False
if not math:
    passing = False or bonus
if not math:
    passing = False or bonus
```

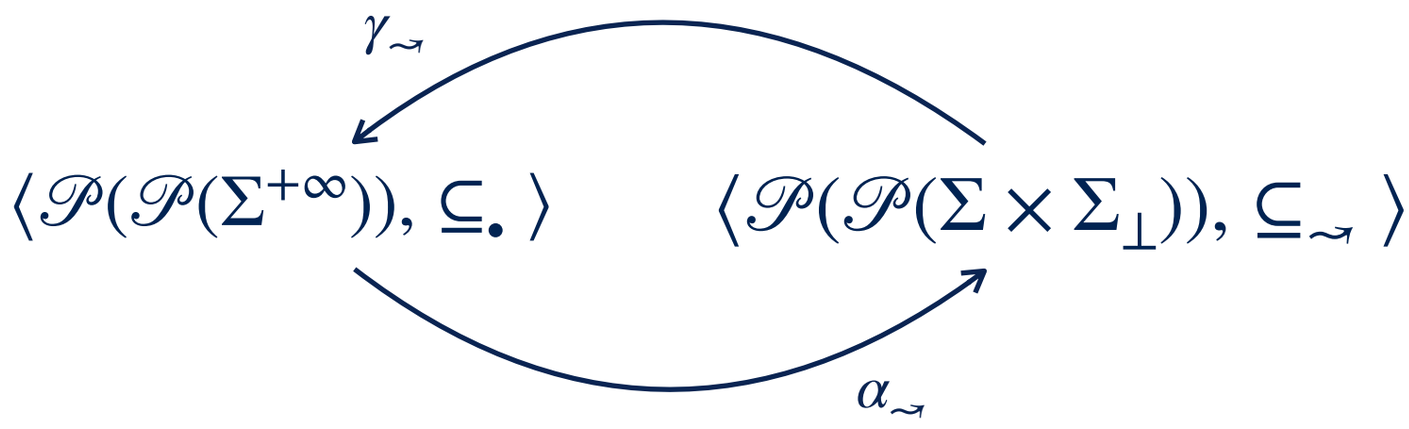


$[\![P]\!]_{\rightsquigarrow}$





Dependency Semantics



$$\alpha_{\rightsquigarrow}(S) \stackrel{\text{def}}{=} \{\{\langle t_0, t_\omega \rangle \in \Sigma \times \Sigma_\perp \mid t \in T\} \mid T \in S\}$$

$$\gamma_{\rightsquigarrow}(S) \stackrel{\text{def}}{=} \{T \in \mathscr{P}(\Sigma^{+\infty}) \mid \{\langle t_0, t_\omega \rangle \in \Sigma \times \Sigma_\perp \mid t \in T\} \in S\}$$

$$[\![P]\!]_{\rightsquigarrow} \stackrel{\text{def}}{=} \alpha_{\rightsquigarrow}([\![P]\!]_{\bullet}) = \{\{\langle t_0, t_\omega \rangle \in \Sigma \times \Sigma \mid t \in [\![P]\!] \cap O\} \mid O \in \mathbb{O}\}$$

Dependency Semantics

Outcome Semantics

Theorem 1. *The outcome semantics $\Lambda_\bullet \in \mathcal{P}(\mathcal{P}(\Sigma^{+\infty}))$ can be expressed as a least fixpoint in $\langle \mathcal{P}(\mathcal{P}(\Sigma^{+\infty})), \sqsubseteq, \sqcup, \sqcap, \{\Sigma^\omega, \emptyset\}, \{\emptyset, \Sigma^+\}\rangle$ as:*

$$\Lambda_\bullet = \mathrm{lfp}^{\sqsubseteq}\ \Theta_\bullet$$

$$\Theta_\bullet(S) \stackrel{def}{=} \{\Omega_{o_1=v_1,\ldots,o_k=v_k} \mid v_1,\ldots,v_k \in \mathrm{V}\} \uplus \{\tau\,;T \mid T \in S\} \qquad (9)$$

where $S_1 \uplus S_2 \stackrel{def}{=} \{S_{1\,o_1=v_1,\ldots,o_k=v_k}^{+} \mid v_1,\ldots,v_k \in \mathrm{V}\} \cup S_1^\omega \cup S_2^\omega$.

$$S_1 \sqsubseteq S_2 \stackrel{\text{def}}{=} \bigwedge_{v_1,\ldots,v_k \in \mathrm{V}} S_{1\,o_1=v_1,\ldots,o_k=v_k}^{+} \subseteq S_{2\,o_1=v_1,\ldots,o_k=v_k}^{+} \wedge S$$

(Urban18])

the outcome semantics $\Lambda_\bullet$ can be equivalently expressed as follows:

$$\Lambda_\bullet = \Lambda_\bullet^+ \cup \Lambda_\bullet^\omega = \mathrm{lfp}_{\emptyset}^{\sqsubseteq}\ \Theta_\bullet^+ \cup \mathrm{lfp}_{\{\Sigma^\omega\}}^{\sqsubseteq}\ \Theta_\bullet^\omega$$

$$\Theta_\bullet^+(S) \stackrel{\text{def}}{=} \{\Omega_{o_1=v_1,\ldots,o_k=v_k} \mid v_1,\ldots,v_k \in \mathrm{V}\} \uplus \{\tau\,;T \mid T \in S\} \qquad (13)$$

$$\Theta_\bullet^\omega(S) \stackrel{\text{def}}{=} \{\tau\,;T \mid T \in S\}$$

Lemma 2. *The abstraction $\Lambda_{\rightsquigarrow}^+ \stackrel{def}{=} \alpha_{\rightsquigarrow}(\Lambda_\bullet^+) \in \mathcal{P}(\mathcal{P}(\Sigma \times \Sigma))$ can be expressed as a least fixpoint in $\langle \mathcal{P}(\mathcal{P}(\Sigma \times \Sigma_\perp)), \sqsubseteq, \sqcup, \sqcap, \{\Sigma \times \{\perp\}, \emptyset\}, \{\emptyset, \Sigma \times \Sigma\}\rangle$ as:*

$$\Lambda_{\rightsquigarrow}^+ = \mathrm{lfp}_{\{\emptyset\}}^{\sqsubseteq}\ \Theta_{\rightsquigarrow}^+$$

$$\Theta_{\rightsquigarrow}^+(S) \stackrel{def}{=} \{\Omega_{o_1=v_1,\ldots,o_k=v_k} \times \Omega_{o_1=v_1,\ldots,o_k=v_k} \mid v_1,\ldots,v_k \in \mathrm{V}\} \uplus \{\tau \circ R \mid R \in S\}$$

Proof (Sketch). By

Lemma 3. *The abstraction $\Lambda_{\rightsquigarrow}^\omega \stackrel{def}{=} \alpha_{\rightsquigarrow}(\Lambda_\bullet^\omega) \in \mathcal{P}(\mathcal{P}(\Sigma \times \Sigma))$ can be expressed as a least fixpoint in $\langle \mathcal{P}(\mathcal{P}(\Sigma \times \Sigma_\perp)), \sqsubseteq, \sqcup, \sqcap, \{\Sigma \times \{\perp\}, \emptyset\}, \{\emptyset, \Sigma \times \Sigma\}\rangle$ as:*

$$\Lambda_{\rightsquigarrow}^\omega = \mathrm{lfp}_{\{\Sigma\times\{\perp\}\}}^{\sqsubseteq}\ \Theta_{\rightsquigarrow}^\omega$$

$$\Theta_{\rightsquigarrow}^\omega(S) \stackrel{def}{=} \{\tau \circ R \mid R \in S\} \qquad (15)$$

Proof (Sketch). By Tarskian fixpoint transfer (cf. Theorem 18 in [12]). □

Dependency Semantics

Lemma 2. *The abstraction* $\Lambda^{+}_{\rightsquigarrow} \stackrel{def}{=} \alpha_{\rightsquigarrow}(\Lambda^{+}_{\bullet}) \in \mathcal{P}\left(\mathcal{P}\left(\Sigma \times \Sigma\right)\right)$ *can be expressed as a least fixpoint in* $\langle \mathcal{P}\left(\mathcal{P}\left(\Sigma \times \Sigma_{\perp}\right)\right), \sqsubseteq, \sqcup, \sqcap, \{\Sigma \times \{\perp\}, \emptyset\}, \{\emptyset, \Sigma \times \Sigma\}\rangle$ *as:*

$$\Lambda^{+}_{\rightsquigarrow} = \mathrm{lfp}^{\dot{\sqsubseteq}}_{\{\emptyset\}} \Theta^{+}_{\rightsquigarrow}$$

$$\Theta^{+}_{\rightsquigarrow}(S) \stackrel{def}{=} \{\Omega_{o_1=v_1,\ldots,o_k=v_k} \times \Omega_{o_1=v_1,\ldots,o_k=v_k} \mid v_1, \ldots, v_k \in \mathrm{V}\} \uplus \{\tau \circ R \mid R \in S\}$$

Proof (Sketch). By

Lemma 3. *The abstraction* $\Lambda^{\omega}_{\rightsquigarrow} \stackrel{def}{=} \alpha_{\rightsquigarrow}(\Lambda^{\omega}_{\bullet}) \in \mathcal{P}\left(\mathcal{P}\left(\Sigma \times \Sigma\right)\right)$ *can be expressed as a least fixpoint in* $\langle \mathcal{P}\left(\mathcal{P}\left(\Sigma \times \Sigma_{\perp}\right)\right), \sqsubseteq, \sqcup, \sqcap, \{\Sigma \times \{\perp\}, \emptyset\}, \{\emptyset, \Sigma \times \Sigma\}\rangle$ *as:*

$$\Lambda^{\omega}_{\rightsquigarrow} = \mathrm{lfp}^{\dot{\sqsubseteq}}_{\{\Sigma \times \{\perp\}\}} \Theta^{\omega}_{\rightsquigarrow}$$

$$\Theta^{\omega}_{\rightsquigarrow}(S) \stackrel{def}{=} \{\tau \circ R \mid R \in S\} \tag{15}$$

Proof (Sketch). By Tarskian fixpoint transfer (cf. Theorem 18 in [12]). □

Theorem 3. *The dependency semantics* $\Lambda_{\rightsquigarrow} \in \mathcal{P}\left(\mathcal{P}\left(\Sigma \times \Sigma_{\perp}\right)\right)$ *can be expressed as a least fixpoint in* $\langle \mathcal{P}\left(\mathcal{P}\left(\Sigma \times \Sigma_{\perp}\right)\right), \sqsubseteq, \sqcup, \sqcap, \{\Sigma \times \{\perp\}, \emptyset\}, \{\emptyset, \Sigma \times \Sigma\}\rangle$ *as:*

$$\Lambda_{\rightsquigarrow} = \Lambda^{+}_{\rightsquigarrow} \cup \Lambda^{\omega}_{\rightsquigarrow} = \mathrm{lfp}^{\dot{\sqsubseteq}}_{\{\Sigma \times \{\perp\}, \emptyset\}} \Theta_{\rightsquigarrow}$$

$$\Theta_{\rightsquigarrow}(S) \stackrel{def}{=} \{\Omega_{o_1=v_1,\ldots,o_k=v_k} \times \Omega_{o_1=v_1,\ldots,o_k=v_k} \mid v_1, \ldots, v_k \in \mathrm{V}\} \uplus \{\tau \circ R \mid R \in S\} \tag{16}$$

Proof (Sketch). The proof follows immediately from Lemma 2 and Lemma 3. □

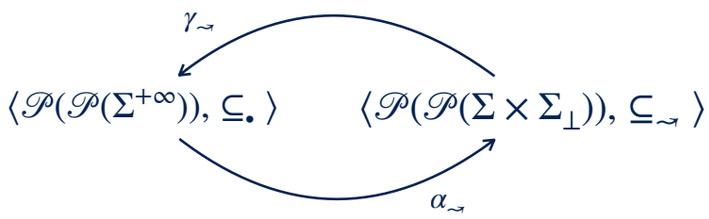
Input Data (Non-)Usage

$$\mathcal{N}_J \stackrel{\text{def}}{=} \{[\![P]\!] \in \mathcal{P}(\Sigma^{+\infty}) \mid \text{UNUSED}_J([\![P]\!])\}$$

$\mathcal{N}_J$ is the set of all programs P (or, rather, their semantics $[\![P]\!]$) that **do not use** the value of the input variables in $J \subseteq I_P$

$$t_0(J) \neq V \Rightarrow \exists t' \in [\![P]\!] : J \Rightarrow t_0(i) = t'_0(i))$$

Dependency Semantics



$$\alpha_{\rightsquigarrow}(S) \stackrel{\text{def}}{=} \{\{\langle t_0, t_\omega \rangle \in \Sigma \times \Sigma_{\perp} \mid t \in T\} \mid T \in S\}$$

$$\gamma_{\rightsquigarrow}(S) \stackrel{\text{def}}{=} \{T \in \mathcal{P}(\Sigma^{+\infty}) \mid \{\langle t_0, t_\omega \rangle \in \Sigma \times \Sigma_{\perp} \mid t \in T\} \in S\}$$

$$[\![P]\!]_{\rightsquigarrow} \stackrel{\text{def}}{=} \alpha_{\rightsquigarrow}([\![P]\!]_{\bullet}) = \{\{\langle t_0, t_\omega \rangle \in \Sigma \times \Sigma \mid t \in [\![P]\!] \cap O\} \mid O \in \mathbb{O}\}$$

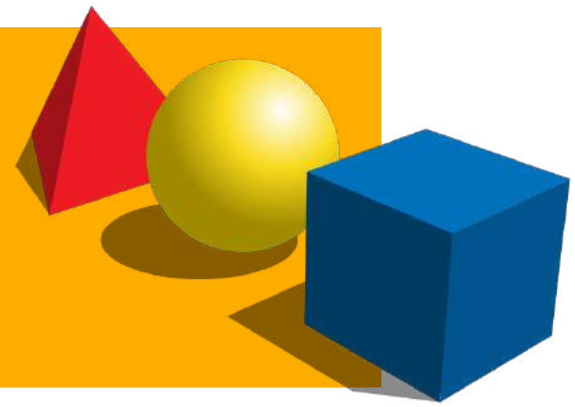
Theorem

$$P \vDash \mathcal{N}_J \Leftrightarrow \{[\![P]\!]\} \subseteq \mathcal{N}_J \Leftrightarrow [\![P]\!]_{\bullet} \subseteq \mathcal{N}_J \Leftrightarrow \gamma_{\rightsquigarrow}([\![P]\!]_{\rightsquigarrow}) \subseteq \mathcal{N}_J$$

Unused Data Analysis

practical tools
targeting specific programs

algorithmic approaches
to decide program properties



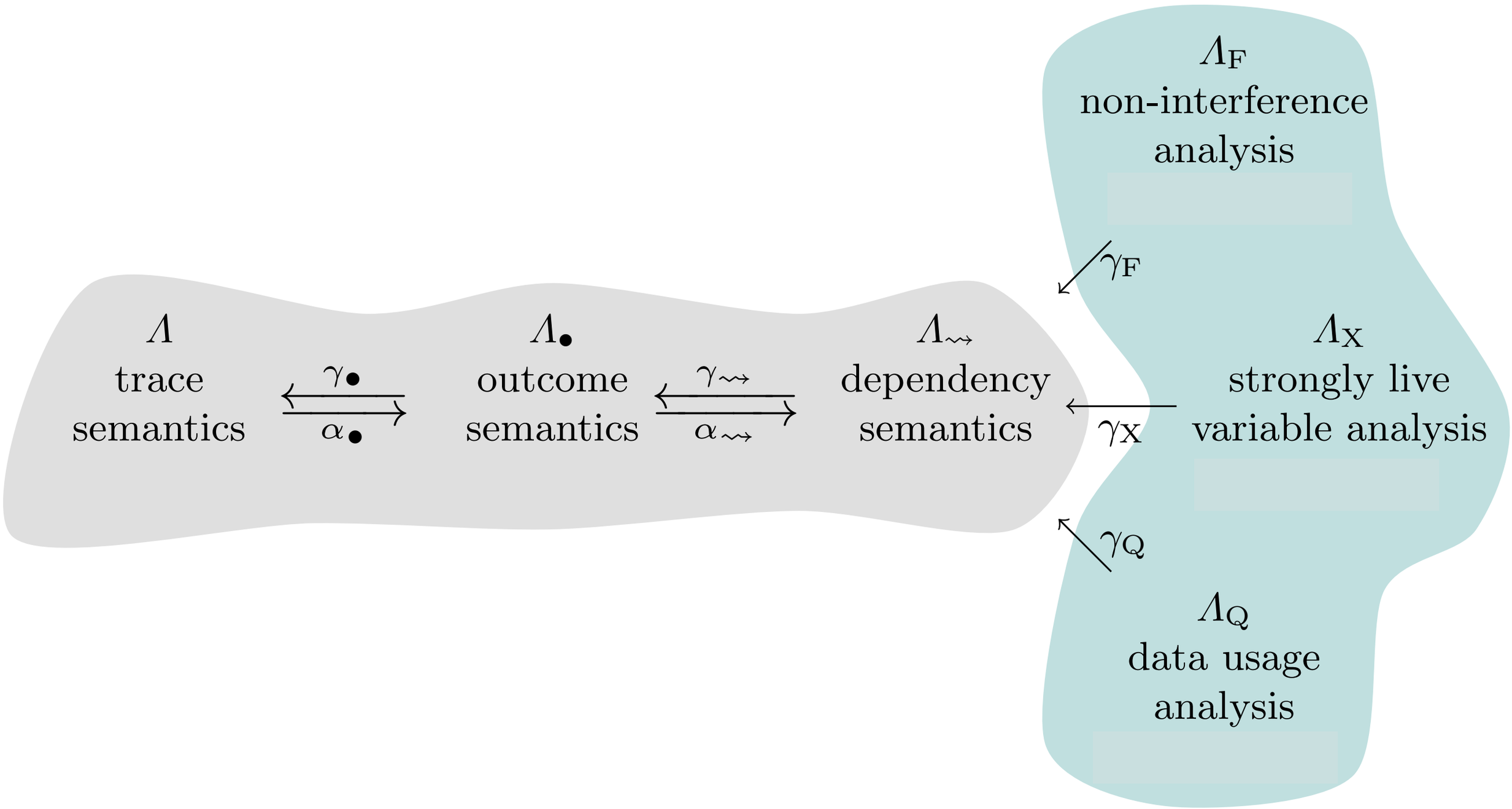
mathematical models
of the program behavior

Input Data (Non-)Usage Abstractions

Over-Approximation of the Used Input Data

$\Rightarrow$ **Under-Approximation** of the Unused Input Data

$$P \vDash \mathcal{N}_{J \not\subseteq J} \Leftarrow \gamma_{\rightsquigarrow}(\gamma_A([\![P]\!]_A)) \subseteq \mathcal{N}_{J \not\subseteq J}$$


Λ
trace semantics
$\gamma_\bullet$
$\alpha_\bullet$
$\Lambda_\bullet$
outcome semantics
$\gamma_{\rightsquigarrow}$
$\alpha_{\rightsquigarrow}$
$\Lambda_{\rightsquigarrow}$
dependency semantics
γ_{X}
Λ_{X}
strongly live variable analysis
γ_{F}
Λ_{F}
non-interference analysis
γ_{Q}
Λ_{Q}
data usage analysis

Secure Information Flow

L ⇢ x

L ⇢ y

H ⇢ t

L ⇢ z

H ⇢ w

possibilistic non-interference coincides with input data (non-)usage when the set J of unused input variables contains *all* input variables:

- **input variables are high-security variables**
- **output variables are low-security variables**

$[\![P]\!]_{\mathrm{F}}$

explicit usage flows

implicit usage flows

$$e ::= v \mid x \mid \texttt{not}\ e \mid e\ \texttt{and}\ e \mid e\ \texttt{or}\ e \qquad \text{(expressions)}$$
$$s ::= \texttt{skip} \mid x = e \mid \texttt{if}\ e\texttt{:}\ s\ \texttt{else:}\ s \mid \texttt{while}\ e\texttt{:}\ s \mid s\ s \qquad \text{(statements)}$$

$$\Theta_{\mathrm{F}}[\![\texttt{skip}]\!](S) \stackrel{\text{def}}{=} S$$

$$\Theta_{\mathrm{F}}[\![x = e]\!](S) \stackrel{\text{def}}{=} \{L \rightsquigarrow y \in S \mid y \neq x\} \cup \{L \rightsquigarrow x \mid \mathcal{V}_{\mathrm{F}}[\![e]\!]S\}$$

$$\Theta_{\mathrm{F}}[\![\texttt{if}\ e\texttt{:}\ s_1\ \texttt{else:}\ s_2]\!](S) \stackrel{\text{def}}{=} \begin{cases} \Theta_{\mathrm{F}}[\![s_1]\!](S) \sqcup_{\mathrm{F}} \Theta_{\mathrm{F}}[\![s_2]\!](S) & \text{if } \mathcal{V}_{\mathrm{F}}[\![e]\!]S \\ \{L \rightsquigarrow x \in S \mid x \notin \mathrm{W}(s_1) \cup \mathrm{W}(s_2)\} & \text{otherwise} \end{cases}$$

$$\Theta_{\mathrm{F}}[\![\texttt{while}\ e\texttt{:}\ s]\!](S) \stackrel{\text{def}}{=} \mathrm{lfp}^{\sqsubseteq_{\mathrm{F}}}_{S}\ \Theta_{\mathrm{F}}[\![\texttt{if}\ e\texttt{:}\ s\ \texttt{else:}\ \texttt{skip}]\!]$$

$$\Theta_{\mathrm{F}}[\![s_1\ s_2]\!](S) \stackrel{\text{def}}{=} \Theta_{\mathrm{F}}[\![s_2]\!] \circ \Theta_{\mathrm{F}}[\![s_1]\!](S)$$

S guarantees a unique value for e independently of values of input variables

$\mathcal{V}_F[\![x]\!]S \Leftrightarrow L \rightsquigarrow x \in S$

set of variables modified by s_i

$\mathcal{L} \stackrel{def}{=} \{L, H\}$: set of security levels

$L \rightsquigarrow x$: dependency constraint

$F \stackrel{def}{=} \{L \rightsquigarrow x \mid x \in X\}$

$\langle \wp(F), \sqsubseteq_F, \sqcup_F \rangle$: abstract domain

$S_1 \sqsubseteq_F S_2 \stackrel{def}{=} S_1 \supseteq S_2$

$S_1 \sqcup_F S_2 \stackrel{def}{=} S_1 \cap S_2$

Hypercollecting Semantics and Its Application to Static Analysis of Information Flow

Mounir Assaf
Stevens Institute of Technology, Hoboken, US
first.last@stevens.edu

David A. Naumann
Stevens Institute of Technology, Hoboken, US
first.last@stevens.edu

Julien Signoles
Software Reliability and Security Lab, CEA LIST, Saclay, FR
first.last@cea.fr

Éric Totel
CIDRE, CentraleSupélec, Rennes, FR
first.last@centralesupelec.fr

Frédéric Tronel
CIDRE, CentraleSupélec, Rennes, FR
first.last@centralesupelec.fr

program is correct if all its traces satisfy the predicate. By c
with such *trace properties*, extensional definitions of deper
involve more than one trace. To express that the final va
depend only on the initial value of a variab
interference in the security
traces with

Secure Information Flow

L ⇢ x

L ⇢ y

H ⇢ t

L ⇢ z

H ⇢ w

possibilistic non-interference coincides with input data (non-)usage when the set J of unused input variables contains *all* input variables:

- **input variables are high-security variables**
- **output variables are low-security variables**

$[\![P]\!]_{\mathrm{F}}$

$$
\begin{aligned}
e &::= v \mid x \mid \texttt{not}\ e \mid e\ \texttt{and}\ e \mid e\ \texttt{or}\ e && \text{(expressions)}\\
s &::= \texttt{skip} \mid x = e \mid \texttt{if}\ e\texttt{:}\ s\ \texttt{else:}\ s \mid \texttt{while}\ e\texttt{:}\ s \mid s\ s && \text{(statements)}
\end{aligned}
$$

$$
\begin{aligned}
\Theta_{\mathrm{F}}[\![\texttt{skip}]\!](S) &\stackrel{\text{def}}{=} S\\
\Theta_{\mathrm{F}}[\![x = e]\!](S) &\stackrel{\text{def}}{=} \{L \rightsquigarrow y \in S \mid y \neq x\} \cup \{L \rightsquigarrow x \mid \mathcal{V}_{\mathrm{F}}[\![e]\!]S\}\\
\Theta_{\mathrm{F}}[\![\texttt{if}\ e\texttt{:}\ s_1\ \texttt{else:}\ s_2]\!](S) &\stackrel{\text{def}}{=} \begin{cases} \Theta_{\mathrm{F}}[\![s_1]\!](S) \sqcup_{\mathrm{F}} \Theta_{\mathrm{F}}[\![s_2]\!](S) & \text{if } \mathcal{V}_{\mathrm{F}}[\![e]\!]S\\ \{L \rightsquigarrow x \in S \mid x \notin \mathrm{W}(s_1) \cup \mathrm{W}(s_2)\} & \text{otherwise} \end{cases}\\
\Theta_{\mathrm{F}}[\![\texttt{while}\ e\texttt{:}\ s]\!](S) &\stackrel{\text{def}}{=} \mathrm{lfp}^{\sqsubseteq_{\mathrm{F}}}_{S}\ \Theta_{\mathrm{F}}[\![\texttt{if}\ e\texttt{:}\ s\ \texttt{else:}\ \texttt{skip}]\!]\\
\Theta_{\mathrm{F}}[\![s_1\ s_2]\!](S) &\stackrel{\text{def}}{=} \Theta_{\mathrm{F}}[\![s_2]\!] \circ \Theta_{\mathrm{F}}[\![s_1]\!](S)
\end{aligned}
$$

```
passing = True
if not english:
    english = False
if not math:
    passing = False or bonus
if not math:
    passing = False or bonus
```

◀ L ⇢ passing, H ⇢ english, math, science, bonus

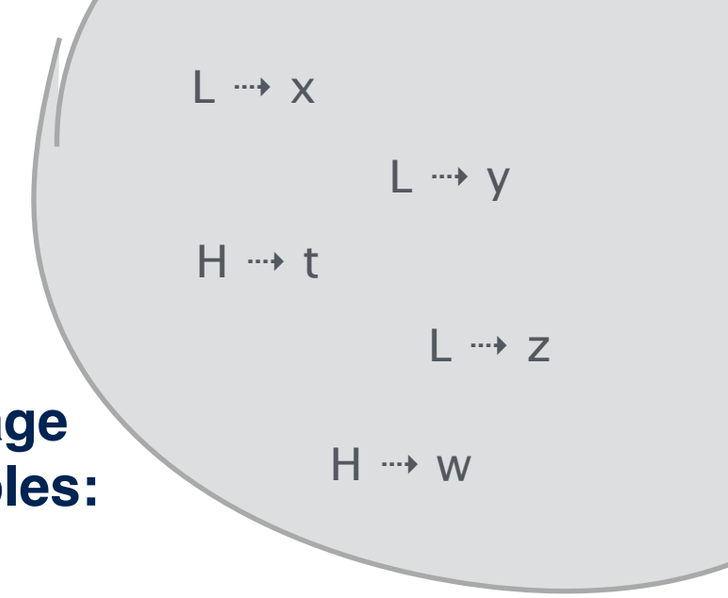
◀ L ⇢ passing, H ⇢ english, math, science, bonus

◀ L ⇢ passing, H ⇢ english, math, science, bonus

◀ H ⇢ english, math, science, bonus, passing

◀ H ⇢ english, math, science, bonus, passing

Secure Information Flow



possibilistic non-interference coincides with input data (non-)usage when the set J of unused input variables contains *all* input variables:

- **input variables are high-security variables**
- **output variables are low-security variables**

and the program is terminating

$[\![P]\!]_{\mathrm{F}}$

$$e ::= v \mid x \mid \texttt{not}\ e \mid e\ \texttt{and}\ e \mid e\ \texttt{or}\ e \qquad \text{(expressions)}$$
$$s ::= \texttt{skip} \mid x = e \mid \texttt{if}\ e\texttt{:}\ s\ \texttt{else:}\ s \mid \texttt{while}\ e\texttt{:}\ s \mid s\ s \qquad \text{(statements)}$$

$$\Theta_{\mathrm{F}}[\![\texttt{skip}]\!](S) \stackrel{\text{def}}{=} S$$

$$\Theta_{\mathrm{F}}[\![x = e]\!](S) \stackrel{\text{def}}{=} \{L \rightsquigarrow y \in S \mid y \neq x\} \cup \{L \rightsquigarrow x \mid \mathcal{V}_{\mathrm{F}}[\![e]\!]S\}$$

$$\Theta_{\mathrm{F}}[\![\texttt{if}\ e\texttt{:}\ s_1\ \texttt{else:}\ s_2]\!](S) \stackrel{\text{def}}{=} \begin{cases} \Theta_{\mathrm{F}}[\![s_1]\!](S) \sqcup_{\mathrm{F}} \Theta_{\mathrm{F}}[\![s_2]\!](S) & \text{if } \mathcal{V}_{\mathrm{F}}[\![e]\!]S \\ \{L \rightsquigarrow x \in S \mid x \notin \mathrm{W}(s_1) \cup \mathrm{W}(s_2)\} & \text{otherwise} \end{cases}$$

$$\Theta_{\mathrm{F}}[\![\texttt{while}\ e\texttt{:}\ s]\!](S) \stackrel{\text{def}}{=} \mathrm{lfp}^{\sqsubseteq_{\mathrm{F}}}_{S}\ \Theta_{\mathrm{F}}[\![\texttt{if}\ e\texttt{:}\ s\ \texttt{else:}\ \texttt{skip}]\!]$$

$$\Theta_{\mathrm{F}}[\![s_1\ s_2]\!](S) \stackrel{\text{def}}{=} \Theta_{\mathrm{F}}[\![s_2]\!] \circ \Theta_{\mathrm{F}}[\![s_1]\!](S)$$

```
passing = True
while not english:
    english = False
```

◀ L ⇢ passing, H ⇢ english, math, science, bonus
◀ L ⇢ passing, H ⇢ english, math, science, bonus
◀ L ⇢ passing, H ⇢ english, math, science, bonus

Theorem

$$P \vDash \mathcal{N}^{*}_{I_P} \Leftarrow \gamma_{\rightsquigarrow}(\gamma_F([\![P]\!]_F)) \subseteq \mathcal{N}^{*}_{I_P}$$

Strong-Liveness

a variable is strongly live if

- **it is used in an assignment to another strongly live variable**
- **it is used in a statement other than an assignment**

$[\![P]\!]_X$

$$e ::= v \mid x \mid \texttt{not}\ e \mid e\ \texttt{and}\ e \mid e\ \texttt{or}\ e \qquad \text{(expressions)}$$
$$s ::= \texttt{skip} \mid x = e \mid \texttt{if}\ e\texttt{:}\ s\ \texttt{else:}\ s \mid \texttt{while}\ e\texttt{:}\ s \mid s\ s \qquad \text{(statements)}$$

$$\Theta_X[\![\texttt{skip}]\!](S) \stackrel{\text{def}}{=} S$$

$$\Theta_X[\![x = e]\!](S) \stackrel{\text{def}}{=} \begin{cases} (S \setminus \{x\}) \cup \text{VARS}(e) & x \in S \\ S & \text{otherwise} \end{cases}$$

$$\Theta_X[\![\texttt{if}\ b\texttt{:}\ s_1\ \texttt{else:}\ s_2]\!](S) \stackrel{\text{def}}{=} \text{VARS}(b) \cup \Theta_X[\![s_1]\!](S) \cup \Theta_X[\![s_2]\!](S)$$

$$\Theta_X[\![\texttt{while}\ b\texttt{:}\ s]\!](S) \stackrel{\text{def}}{=} \text{VARS}(b) \cup \Theta_X[\![s]\!](S)$$

$$\Theta_X[\![s_1\ s_2]\!](S) \stackrel{\text{def}}{=} \Theta_X[\![s_1]\!] \circ \Theta_X[\![s_2]\!](S)$$

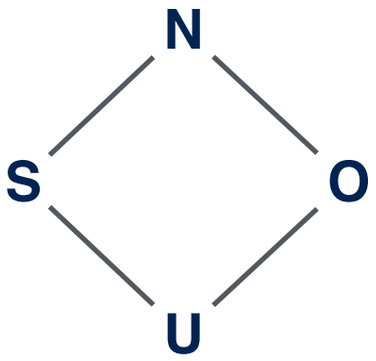
Theorem

$$P \vDash \mathcal{N}_J \Leftarrow \gamma_{\rightsquigarrow}(\gamma_X([\![P]\!]_X)) \subseteq \mathcal{N}_J$$

```
passing = True
if not english:
    english = False
if not math:
    passing = False or bonus
if not math:
    passing = False or bonus
```

◀ { bonus, math, english }
◀ { bonus, math, english }
◀ { bonus, math }
◀ { bonus, math }
◀ { bonus, math }
◀ { bonus, math }
◀ { bonus }
◀ { passing }

Syntactic (Non-)Usage



- **U**: used in the current scope (or an inner scope)
- **S**: used in an outer scope
- **O**: used in an outer scope and overridden in the current scope
- **N**: not used

x ⇢ U

y ⇢ S | y ⇢ U

t ⇢ N

z ⇢ N

w ⇢ O | w ⇢ U

$[\![P]\!]_U$

$$\Theta_Q[\![\texttt{skip}]\!](q) \stackrel{\text{def}}{=} q$$

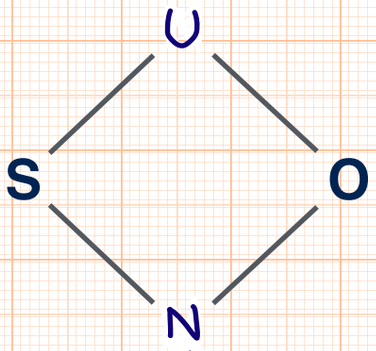
$$\Theta_Q[\![x = e]\!](q) \stackrel{\text{def}}{=} \text{ASSIGN}[\![x = e]\!](q)$$

$$\Theta_Q[\![\texttt{if } b\texttt{: } s_1 \texttt{ else: } s_2]\!](q) \stackrel{\text{def}}{=} \text{POP} \circ \text{FILTER}[\![b]\!] \circ \Theta_Q[\![s_1]\!] \circ \text{PUSH}(q)$$
$$\sqcup_Q \text{POP} \circ \text{FILTER}[\![b]\!] \circ \Theta_Q[\![s_2]\!] \circ \text{PUSH}(q)$$

$$\Theta_Q[\![\texttt{while } b\texttt{: } s]\!](q) \stackrel{\text{def}}{=} \text{lfp}_t^{\sqsubseteq_Q}\ \Theta_Q[\![\texttt{if } b\texttt{: } s \texttt{ else: skip}]\!]$$

$$\Theta_Q[\![s_1\ s_2]\!](q) \stackrel{\text{def}}{=} \Theta_Q[\![s_1]\!] \circ \Theta_Q[\![s_2]\!](q)$$

Syntactic (Non-)Usage



- **U**: used in the current scope (or an inner scope)
- **S**: used in an outer scope
- **O**: used in an outer scope and overridden in the current scope
- **N**: not used

x ⇢ U

y ⇢ S | y ⇢ U

t ⇢ N

z ⇢ N

w ⇢ O | w ⇢ U

$[\![P]\!]_U$

$$\Theta_Q[\![\texttt{skip}]\!](q) \stackrel{\text{def}}{=} q$$

$$\Theta_Q[\![x = e]\!](q) \stackrel{\text{def}}{=} \text{ASSIGN}[\![x = e]\!](q)$$

$$\Theta_Q[\![\texttt{if } b\texttt{: } s_1 \texttt{ else: } s_2]\!](q) \stackrel{\text{def}}{=} \text{POP} \circ \text{FILTER}[\![b]\!] \circ \Theta_Q[\![s_1]\!] \circ \text{PUSH}(q) \sqcup_Q \text{POP} \circ \text{FILTER}[\![b]\!] \circ \Theta_Q[\![s_2]\!] \circ \text{PUSH}(q)$$

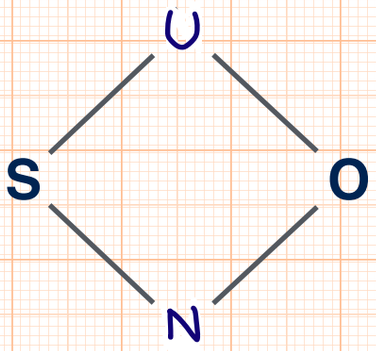
$$\Theta_Q[\![\texttt{while } b\texttt{: } s]\!](q) \stackrel{\text{def}}{=} \text{lfp}_t^{\sqsubseteq_Q}\ \Theta_Q[\![\texttt{if } b\texttt{: } s \texttt{ else: skip}]\!]$$

$$\Theta_Q[\![s_1\ s_2]\!](q) \stackrel{\text{def}}{=} \Theta_Q[\![s_1]\!] \circ \Theta_Q[\![s_2]\!](q)$$

```
passing = True
if not english:
    english = False

if not math:
    passing = False or bonus

if not math:
    passing = False or bonus
```

passing ⇢ U (any other variable maps to N)

Syntactic (Non-)Usage



- **U**: used in the current scope (or an inner scope)
- **S**: used in an outer scope
- **O**: used in an outer scope and overridden in the current scope
- **N**: not used

x ⇢ U

y ⇢ S | y ⇢ U

t ⇢ N

z ⇢ N

w ⇢ O | w ⇢ U

$[\![P]\!]_U$

```
passing = True
if not english:
    english = False

if not math:
    passing = False or bonus

if not math:
    passing = False or bonus
```

$$\Theta_Q[\![\texttt{skip}]\!](q) \stackrel{\text{def}}{=} q$$

$$\Theta_Q[\![x = e]\!](q) \stackrel{\text{def}}{=} \text{ASSIGN}[\![x = e]\!](q)$$

$$\Theta_Q[\![\texttt{if } b\texttt{: } s_1 \texttt{ else: } s_2]\!](q) \stackrel{\text{def}}{=} \text{POP} \circ \text{FILTER}[\![b]\!] \circ \Theta_Q[\![s_1]\!] \circ \text{PUSH}(q) \sqcup_Q \text{POP} \circ \text{FILTER}[\![b]\!] \circ \Theta_Q[\![s_2]\!] \circ \text{PUSH}(q)$$

$$\Theta_Q[\![\texttt{while } b\texttt{: } s]\!](q) \stackrel{\text{def}}{=} \text{lfp}_t^{\sqsubseteq_Q}\ \Theta_Q[\![\texttt{if } b\texttt{: } s \texttt{ else: skip}]\!]$$

$$\Theta_Q[\![s_1\ s_2]\!](q) \stackrel{\text{def}}{=} \Theta_Q[\![s_1]\!] \circ \Theta_Q[\![s_2]\!](q)$$

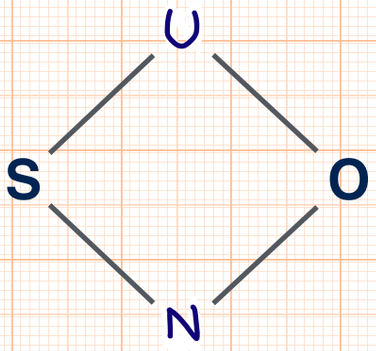
x → U ⇒ x → S
x → O ⇒ x → N

passing ⇢ S | passing ⇢ U

passing ⇢ U

domain elements are stacks of maps matching nesting level of analyzed statements

Syntactic (Non-)Usage



- **U**: used in the current scope (or an inner scope)
- **S**: used in an outer scope
- **O**: used in an outer scope and overridden in the current scope
- **N**: not used

x ⇢ U

y ⇢ S | y ⇢ U

t ⇢ N

z ⇢ N

w ⇢ O | w ⇢ U

$[\![P]\!]_U$

```
passing = True

if not english:

    english = False


if not math:

    passing = False or bonus


if not math:
    passing = False or bonus
```

$$\Theta_Q[\![\texttt{skip}]\!](q) \stackrel{\text{def}}{=} q$$

$$\Theta_Q[\![x = e]\!](q) \stackrel{\text{def}}{=} \text{ASSIGN}[\![x = e]\!](q)$$

$$\Theta_Q[\![\texttt{if } b\texttt{: } s_1 \texttt{ else: } s_2]\!](q) \stackrel{\text{def}}{=} \text{POP} \circ \text{FILTER}[\![b]\!] \circ \Theta_Q[\![s_1]\!] \circ \text{PUSH}(q) \sqcup_Q \text{POP} \circ \text{FILTER}[\![b]\!] \circ \Theta_Q[\![s_2]\!] \circ \text{PUSH}(q)$$

$$\Theta_Q[\![\texttt{while } b\texttt{: } s]\!](q) \stackrel{\text{def}}{=} \text{lfp}_t^{\sqsubseteq_Q}\ \Theta_Q[\![\texttt{if } b\texttt{: } s \texttt{ else: skip}]\!]$$

$$\Theta_Q[\![s_1\ s_2]\!](q) \stackrel{\text{def}}{=} \Theta_Q[\![s_1]\!] \circ \Theta_Q[\![s_2]\!](q)$$

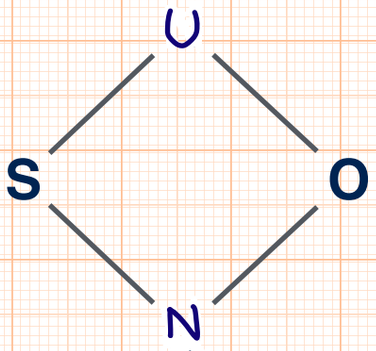
if the assigned variable was used (U or S) it becomes overwritten (O) if not also used in the expression being assigned; otherwise it becomes freshly used (U)

bonus ⇢ U, passing ⇢ O | passing ⇢ U

passing ⇢ S | passing ⇢ U

passing ⇢ U

Syntactic (Non-)Usage



- **U**: used in the current scope (or an inner scope)
- **S**: used in an outer scope
- **O**: used in an outer scope and overridden in the current scope
- **N**: not used

x ⇢ U

y ⇢ S | y ⇢ U

t ⇢ N

z ⇢ N

w ⇢ O | w ⇢ U

$[\![P]\!]_U$

$$\Theta_Q[\![\texttt{skip}]\!](q) \stackrel{\text{def}}{=} q$$

$$\Theta_Q[\![x = e]\!](q) \stackrel{\text{def}}{=} \text{ASSIGN}[\![x = e]\!](q)$$

$$\Theta_Q[\![\texttt{if } b\texttt{: } s_1 \texttt{ else: } s_2]\!](q) \stackrel{\text{def}}{=} \text{POP} \circ \text{FILTER}[\![b]\!] \circ \Theta_Q[\![s_1]\!] \circ \text{PUSH}(q) \sqcup_Q \text{POP} \circ \text{FILTER}[\![b]\!] \circ \Theta_Q[\![s_2]\!] \circ \text{PUSH}(q)$$

$$\Theta_Q[\![\texttt{while } b\texttt{: } s]\!](q) \stackrel{\text{def}}{=} \text{lfp}_t^{\sqsubseteq_Q} \Theta_Q[\![\texttt{if } b\texttt{: } s \texttt{ else: skip}]\!]$$

$$\Theta_Q[\![s_1\ s_2]\!](q) \stackrel{\text{def}}{=} \Theta_Q[\![s_1]\!] \circ \Theta_Q[\![s_2]\!](q)$$

a variable becomes used (U)
if it appears in the boolean condition
of a statement that uses (U) or modifies (O)
another variable

```
passing = True

if not english:

    english = False


if not math:

    passing = False or bonus
```

```
if not math:
    passing = False or bonus
```

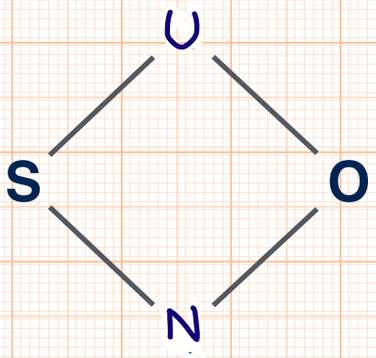
math, bonus, passing ⇢ U

bonus ⇢ U, passing ⇢ O | passing ⇢ U

passing ⇢ S | passing ⇢ U

passing ⇢ U

Syntactic (Non-)Usage



- **U**: used in the current scope (or an inner scope)
- **S**: used in an outer scope
- **O**: used in an outer scope and overridden in the current scope
- **N**: not used

x ⇢ U

y ⇢ S | y ⇢ U

t ⇢ N

z ⇢ N

w ⇢ O | w ⇢ U

$[\![P]\!]_U$

$$\Theta_Q[\![\texttt{skip}]\!](q) \stackrel{\text{def}}{=} q$$

$$\Theta_Q[\![x = e]\!](q) \stackrel{\text{def}}{=} \text{ASSIGN}[\![x = e]\!](q)$$

$$\Theta_Q[\![\texttt{if } b\texttt{: } s_1 \texttt{ else: } s_2]\!](q) \stackrel{\text{def}}{=} \text{POP} \circ \text{FILTER}[\![b]\!] \circ \Theta_Q[\![s_1]\!] \circ \text{PUSH}(q) \sqcup_Q \text{POP} \circ \text{FILTER}[\![b]\!] \circ \Theta_Q[\![s_2]\!] \circ \text{PUSH}(q)$$

$$\Theta_Q[\![\texttt{while } b\texttt{: } s]\!](q) \stackrel{\text{def}}{=} \text{lfp}_t^{\sqsubseteq_Q}\ \Theta_Q[\![\texttt{if } b\texttt{: } s \texttt{ else: skip}]\!]$$

$$\Theta_Q[\![s_1\ s_2]\!](q) \stackrel{\text{def}}{=} \Theta_Q[\![s_1]\!] \circ \Theta_Q[\![s_2]\!](q)$$

restores the previous value of variables (before increasing the nesting level) if it has not changed since

```
passing = True
if not english:
    english = False

if not math:
    passing = False or bonus

if not math:
    passing = False or bonus
```

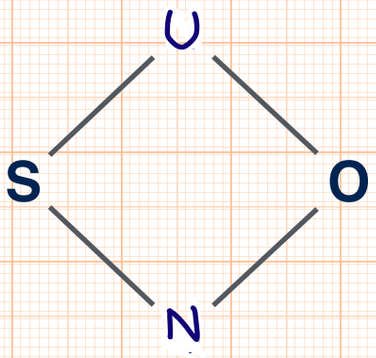
math, bonus, passing ⇢ U
math, bonus ↦ U, passing ↦ O

bonus ⇢ U, passing ⇢ O | passing ⇢ U

passing ⇢ S | passing ⇢ U

passing ⇢ U

Syntactic (Non-)Usage



- **U**: used in the current scope (or an inner scope)
- **S**: used in an outer scope
- **O**: used in an outer scope and overridden in the current scope
- **N**: not used

x ⇢ U

y ⇢ S | y ⇢ U

t ⇢ N

z ⇢ N

w ⇢ O | w ⇢ U

$[\![P]\!]_U$

$$\Theta_Q[\![\texttt{skip}]\!](q) \stackrel{\text{def}}{=} q$$

$$\Theta_Q[\![x = e]\!](q) \stackrel{\text{def}}{=} \text{ASSIGN}[\![x = e]\!](q)$$

$$\Theta_Q[\![\texttt{if } b\texttt{: } s_1 \texttt{ else: } s_2]\!](q) \stackrel{\text{def}}{=} \text{POP} \circ \text{FILTER}[\![b]\!] \circ \Theta_Q[\![s_1]\!] \circ \text{PUSH}(q) \sqcup_Q \text{POP} \circ \text{FILTER}[\![b]\!] \circ \Theta_Q[\![s_2]\!] \circ \text{PUSH}(q)$$

$$\Theta_Q[\![\texttt{while } b\texttt{: } s]\!](q) \stackrel{\text{def}}{=} \text{lfp}_t^{\sqsubseteq_Q}\ \Theta_Q[\![\texttt{if } b\texttt{: } s \texttt{ else: skip}]\!]$$

$$\Theta_Q[\![s_1\ s_2]\!](q) \stackrel{\text{def}}{=} \Theta_Q[\![s_1]\!] \circ \Theta_Q[\![s_2]\!](q)$$

```
passing = True
if not english:
    english = False

if not math:
    passing = False or bonus
```

```
if not math:
    passing = False or bonus
```

math, bonus, passing ⇢ U

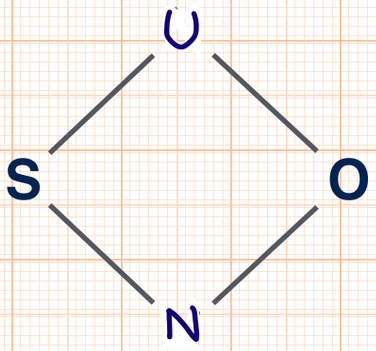
bonus ⇢ U, passing ⇢ O | passing ⇢ U

passing ⇢ S | passing ⇢ U

passing ⇢ U

math, bonus ↦ U, passing ↦ O ⊔Q passing ↦ U

Syntactic (Non-)Usage



- **U**: used in the current scope (or an inner scope)
- **S**: used in an outer scope
- **O**: used in an outer scope and overridden in the current scope
- **N**: not used

x ⇢ U

y ⇢ S | y ⇢ U

t ⇢ N

z ⇢ N

w ⇢ O | w ⇢ U

$[\![P]\!]_U$

$$\Theta_Q[\![\texttt{skip}]\!](q) \stackrel{\text{def}}{=} q$$

$$\Theta_Q[\![x = e]\!](q) \stackrel{\text{def}}{=} \text{ASSIGN}[\![x = e]\!](q)$$

$$\Theta_Q[\![\texttt{if } b\texttt{: } s_1 \texttt{ else: } s_2]\!](q) \stackrel{\text{def}}{=} \text{POP} \circ \text{FILTER}[\![b]\!] \circ \Theta_Q[\![s_1]\!] \circ \text{PUSH}(q)$$
$$\sqcup_Q \text{POP} \circ \text{FILTER}[\![b]\!] \circ \Theta_Q[\![s_2]\!] \circ \text{PUSH}(q)$$

$$\Theta_Q[\![\texttt{while } b\texttt{: } s]\!](q) \stackrel{\text{def}}{=} \text{lfp}_t^{\sqsubseteq_Q}\ \Theta_Q[\![\texttt{if } b\texttt{: } s \texttt{ else: skip}]\!]$$

$$\Theta_Q[\![s_1\ s_2]\!](q) \stackrel{\text{def}}{=} \Theta_Q[\![s_1]\!] \circ \Theta_Q[\![s_2]\!](q)$$

```
passing = True
if not english:
    english = False

if not math:
    passing = False or bonus
if not math:
    passing = False or bonus
```

math, bonus, passing ⇢ S | math, bonus, passing ⇢ U

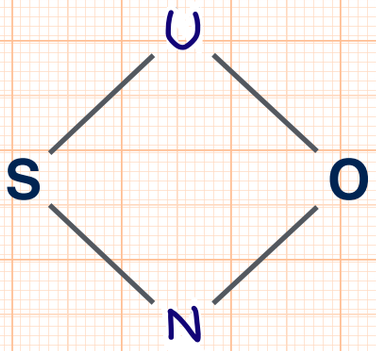
math, bonus, passing ⇢ U

bonus ⇢ U, passing ⇢ O | passing ⇢ U

passing ⇢ S | passing ⇢ U

passing ⇢ U

Syntactic (Non-)Usage



- **U**: used in the current scope (or an inner scope)
- **S**: used in an outer scope
- **O**: used in an outer scope and overridden in the current scope
- **N**: not used

x ⇢ U

y ⇢ S | y ⇢ U

t ⇢ N

z ⇢ N

w ⇢ O | w ⇢ U

$[\![P]\!]_U$

$$\Theta_Q[\![\texttt{skip}]\!](q) \stackrel{\text{def}}{=} q$$

$$\Theta_Q[\![x = e]\!](q) \stackrel{\text{def}}{=} \text{ASSIGN}[\![x = e]\!](q)$$

$$\Theta_Q[\![\texttt{if } b\texttt{: } s_1 \texttt{ else: } s_2]\!](q) \stackrel{\text{def}}{=} \text{POP} \circ \text{FILTER}[\![b]\!] \circ \Theta_Q[\![s_1]\!] \circ \text{PUSH}(q) \sqcup_Q \text{POP} \circ \text{FILTER}[\![b]\!] \circ \Theta_Q[\![s_2]\!] \circ \text{PUSH}(q)$$

$$\Theta_Q[\![\texttt{while } b\texttt{: } s]\!](q) \stackrel{\text{def}}{=} \text{lfp}_t^{\sqsubseteq_Q}\ \Theta_Q[\![\texttt{if } b\texttt{: } s \texttt{ else: skip}]\!]$$

$$\Theta_Q[\![s_1\ s_2]\!](q) \stackrel{\text{def}}{=} \Theta_Q[\![s_1]\!] \circ \Theta_Q[\![s_2]\!](q)$$

```
passing = True

if not english:

    english = False


if not math:
    passing = False or bonus
if not math:
    passing = False or bonus
```

math ⇢ S, bonus ⇢ U, passing ⇢ O | …

math, bonus, passing ⇢ S | math, bonus, passing ⇢ U

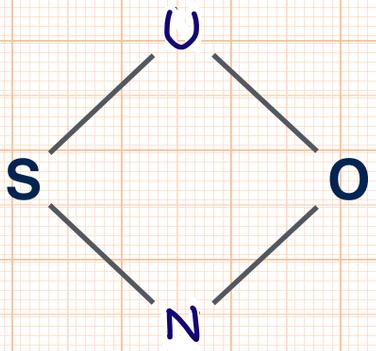
math, bonus, passing ⇢ U

bonus ⇢ U, passing ⇢ O | passing ⇢ U

passing ⇢ S | passing ⇢ U

passing ⇢ U

Syntactic (Non-)Usage



- **U**: used in the current scope (or an inner scope)
- **S**: used in an outer scope
- **O**: used in an outer scope and overridden in the current scope
- **N**: not used

x ⇢ U

y ⇢ S | y ⇢ U

t ⇢ N

z ⇢ N

w ⇢ O | w ⇢ U

$[\![P]\!]_U$

$$\Theta_Q[\![\texttt{skip}]\!](q) \stackrel{\text{def}}{=} q$$

$$\Theta_Q[\![x = e]\!](q) \stackrel{\text{def}}{=} \text{ASSIGN}[\![x = e]\!](q)$$

$$\Theta_Q[\![\texttt{if } b\texttt{: } s_1 \texttt{ else: } s_2]\!](q) \stackrel{\text{def}}{=} \text{POP} \circ \text{FILTER}[\![b]\!] \circ \Theta_Q[\![s_1]\!] \circ \text{PUSH}(q) \sqcup_Q \text{POP} \circ \text{FILTER}[\![b]\!] \circ \Theta_Q[\![s_2]\!] \circ \text{PUSH}(q)$$

$$\Theta_Q[\![\texttt{while } b\texttt{: } s]\!](q) \stackrel{\text{def}}{=} \text{lfp}_t^{\sqsubseteq_Q}\ \Theta_Q[\![\texttt{if } b\texttt{: } s \texttt{ else: skip}]\!]$$

$$\Theta_Q[\![s_1\ s_2]\!](q) \stackrel{\text{def}}{=} \Theta_Q[\![s_1]\!] \circ \Theta_Q[\![s_2]\!](q)$$

passing = **True**

if not english:

english = **False**

if not math:

passing = **False or** bonus

if not math:

passing = **False or** bonus

math, bonus, passing ⇢ U

math ⇢ S, bonus ⇢ U, passing ⇢ O | …

math, bonus, passing ⇢ S | math, bonus, passing ⇢ U

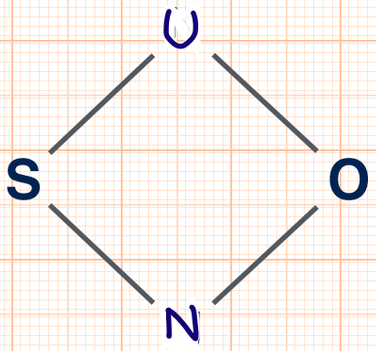
math, bonus, passing ⇢ U

bonus ⇢ U, passing ⇢ O | passing ⇢ U

passing ⇢ S | passing ⇢ U

passing ⇢ U

Syntactic (Non-)Usage



- **U**: used in the current scope (or an inner scope)
- **S**: used in an outer scope
- **O**: used in an outer scope and overridden in the current scope
- **N**: not used

x ⇢ U

y ⇢ S | y ⇢ U

t ⇢ N

z ⇢ N

w ⇢ O | w ⇢ U

$[\![P]\!]_U$

$$\Theta_Q[\![\texttt{skip}]\!](q) \stackrel{\text{def}}{=} q$$

$$\Theta_Q[\![x = e]\!](q) \stackrel{\text{def}}{=} \text{ASSIGN}[\![x = e]\!](q)$$

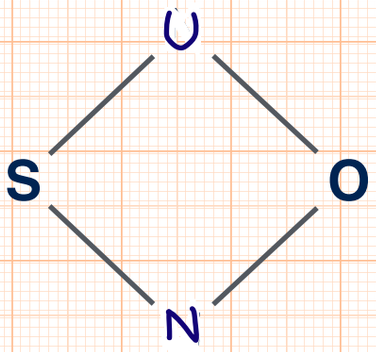
$$\Theta_Q[\![\texttt{if } b\texttt{: } s_1 \texttt{ else: } s_2]\!](q) \stackrel{\text{def}}{=} \text{POP} \circ \text{FILTER}[\![b]\!] \circ \Theta_Q[\![s_1]\!] \circ \text{PUSH}(q) \sqcup_Q \text{POP} \circ \text{FILTER}[\![b]\!] \circ \Theta_Q[\![s_2]\!] \circ \text{PUSH}(q)$$

$$\Theta_Q[\![\texttt{while } b\texttt{: } s]\!](q) \stackrel{\text{def}}{=} \text{lfp}_t^{\sqsubseteq_Q}\ \Theta_Q[\![\texttt{if } b\texttt{: } s \texttt{ else: skip}]\!]$$

$$\Theta_Q[\![s_1\ s_2]\!](q) \stackrel{\text{def}}{=} \Theta_Q[\![s_1]\!] \circ \Theta_Q[\![s_2]\!](q)$$

```
passing = True
if not english:
    english = False
                                  math, bonus, passing ⇢ S | math, bonus, passing ⇢ U
                                  math, bonus, passing ⇢ U
if not math:
                                  math ⇢ S, bonus ⇢ U, passing ⇢ O | ...
    passing = False or bonus
                                  math, bonus, passing ⇢ S | math, bonus, passing ⇢ U
                                  math, bonus, passing ⇢ U
if not math:
                                  bonus ⇢ U, passing ⇢ O | passing ⇢ U
    passing = False or bonus
                                  passing ⇢ S | passing ⇢ U
                                  passing ⇢ U
```

Syntactic (Non-)Usage



- **U**: used in the current scope (or an inner scope)
- **S**: used in an outer scope
- **O**: used in an outer scope and overridden in the current scope
- **N**: not used

x ⇢ U

y ⇢ S | y ⇢ U

t ⇢ N

z ⇢ N

w ⇢ O | w ⇢ U

$[\![P]\!]_U$

the input variables english and science are definetly not used by the program

```
passing = True
if not english:
    english = False
if not math:
    passing = False or bonus
if not math:
    passing = False or bonus
```

math, bonus ⇢ U, passing ⇢ O

math, bonus, passing ⇢ U

math, bonus, passing ⇢ S | math, bonus, passing ⇢ U

math, bonus, passing ⇢ S | math, bonus, passing ⇢ U

math, bonus, passing ⇢ U

passing ⇢ U

$$\Theta_Q[\![\texttt{skip}]\!](q) \stackrel{\text{def}}{=} q$$

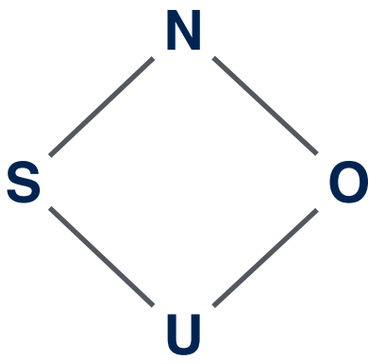
$$\Theta_Q[\![x = e]\!](q) \stackrel{\text{def}}{=} \text{ASSIGN}[\![x = e]\!](q)$$

$$\Theta_Q[\![\texttt{if } b\texttt{: } s_1 \texttt{ else: } s_2]\!](q) \stackrel{\text{def}}{=} \text{POP} \circ \text{FILTER}[\![b]\!] \circ \Theta_Q[\![s_1]\!] \circ \text{PUSH}(q) \sqcup_Q \text{POP} \circ \text{FILTER}[\![b]\!] \circ \Theta_Q[\![s_2]\!] \circ \text{PUSH}(q)$$

$$\Theta_Q[\![\texttt{while } b\texttt{: } s]\!](q) \stackrel{\text{def}}{=} \text{lfp}_t^{\sqsubseteq_Q}\, \Theta_Q[\![\texttt{if } b\texttt{: } s \texttt{ else: skip}]\!]$$

$$\Theta_Q[\![s_1\ s_2]\!](q) \stackrel{\text{def}}{=} \Theta_Q[\![s_1]\!] \circ \Theta_Q[\![s_2]\!](q)$$

Syntactic (Non-)Usage



- **U**: used in the current scope (or an inner scope)
- **S**: used in an outer scope
- **O**: used in an outer scope and overridden in the current scope
- **N**: not used

x ⇢ U

y ⇢ S | y ⇢ U

t ⇢ N

z ⇢ N

w ⇢ O | w ⇢ U

$[\![P]\!]_U$

$$e ::= v \mid x \mid \texttt{not}\ e \mid e\ \texttt{and}\ e \mid e\ \texttt{or}\ e \qquad \text{(expressions)}$$
$$s ::= \texttt{skip} \mid x = e \mid \texttt{if}\ e\texttt{:}\ s\ \texttt{else:}\ s \mid \texttt{while}\ e\texttt{:}\ s \mid s\ s \qquad \text{(statements)}$$

$$\Theta_{\mathrm{Q}}[\![\texttt{skip}]\!](q) \stackrel{\text{def}}{=} q$$
$$\Theta_{\mathrm{Q}}[\![x = e]\!](q) \stackrel{\text{def}}{=} \mathrm{ASSIGN}[\![x = e]\!](q)$$
$$\Theta_{\mathrm{Q}}[\![\texttt{if}\ b\texttt{:}\ s_1\ \texttt{else:}\ s_2]\!](q) \stackrel{\text{def}}{=} \mathrm{POP} \circ \mathrm{FILTER}[\![b]\!] \circ \Theta_{\mathrm{Q}}[\![s_1]\!] \circ \mathrm{PUSH}(q)$$
$$\sqcup_{\mathrm{Q}}\ \mathrm{POP} \circ \mathrm{FILTER}[\![b]\!] \circ \Theta_{\mathrm{Q}}[\![s_2]\!] \circ \mathrm{PUSH}(q)$$
$$\Theta_{\mathrm{Q}}[\![\texttt{while}\ b\texttt{:}\ s]\!](q) \stackrel{\text{def}}{=} \mathrm{lfp}_t^{\sqsubseteq_{\mathrm{Q}}}\ \Theta_{\mathrm{Q}}[\![\texttt{if}\ b\texttt{:}\ s\ \texttt{else:}\ \texttt{skip}]\!]$$
$$\Theta_{\mathrm{Q}}[\![s_1\ s_2]\!](q) \stackrel{\text{def}}{=} \Theta_{\mathrm{Q}}[\![s_1]\!] \circ \Theta_{\mathrm{Q}}[\![s_2]\!](q)$$

```
passing = True
while not english:
    english = False
```

passing ⇢ O

passing ⇢ U

passing ⇢ U

Theorem

$$P \vDash \mathcal{N}_J^* \Leftarrow \gamma_{\rightsquigarrow}(\gamma_Q([\![P]\!]_Q)) \subseteq \mathcal{N}_J^*$$

Piecewise Syntactic (Non-)Usage

{0} N {i} U {i+1} O {len}

{0} N {len}

{0} S {len} | {0} U {len}

$$e ::= v \mid x \mid \texttt{not}\ e \mid e\ \texttt{and}\ e \mid e\ \texttt{or}\ e \mid a[e] \mid \texttt{len}(a) \mid e \oplus e \mid e \bowtie e \quad \text{(expressions)}$$

$$s ::= \texttt{skip} \mid x = e \mid \texttt{if}\ e\texttt{:}\ s\ \texttt{else:}\ s \mid \texttt{while}\ e\texttt{:}\ s \mid s\ s \mid a[e] = e \quad \text{(statements)}$$

$[\![P]\!]_S$

```
grades = list(map(int, input().split()))

count = 0
i = 1
while i < len(grades):
    if grades[i] < 4:
        count = count + 1
    i = i + 1
if 2 * count < len(grades):
    passing = True
else:
    passing = False
print(passing)
```

grades[0] is definitely not used by the program

grades ⇢ {0} N {1}? U {2}? U {len(grades)}?

ERROR: 1 SHOULD BE 0

grades ⇢ {0} N {i}? U {i+1}? U {len(grades)}?

grades ⇢ {0} N {i}? U {i+1}? S {i+2}? S {len(grades)}? | …

grades ⇢ {0} N {i+1}? S {i+2}? S {len(grades)}? | … | …

grades ⇢ {0} N {i+1}? S {i+2}? S {len(grades)}? | …

grades ⇢ {0} N {i}? S {i+1}? S {len(grades)}? | …

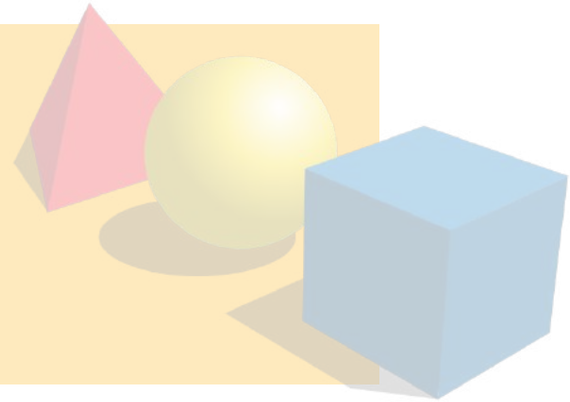
grades ⇢ {0} N {len(grades)}?

grades ⇢ { 0 } N { len(grades) }?

Unused Data Analysis

practical tools
targeting specific programs

algorithmic approaches
to decide program properties



mathematical models
of the program behavior

github.com

caterinaurban / Lyra

Type / to search

Code | Issues 1 | Pull requests 1 | Actions | Projects | Wiki | Security 6 | Insights | Settings

Lyra Public

Unpin | Unwatch 4 | Fork 9 | Star 25

master | 1 branch | 0 tags | Go to file | Add file | Code

caterinaurban update for Python 3.9 — e37b228 on Nov 7 — 1,144 commits

docs	documentation	5 years ago
src	update for Python 3.9	last month
.gitignore	[wip] adding .DS_Store mac file	9 months ago
.travis.yml	added fulara unittests to travis	5 years ago
LICENSE	Initial commit	6 years ago
README.md	Merge pull request #78 from caterinaurban/build-status	5 years ago
icon.png	various	6 years ago
lyra.png	logo	6 years ago
requirements.txt	list creation	5 years ago
setup.py	main file	6 years ago

README.md

Lyra - Static Analyzer for Data Science Applications

About

No description or website provided.

python | data-science | static-analysis | abstract-interpretation

Readme
MPL-2.0 license
Activity
25 stars
4 watching
9 forks

Releases

No releases published
Create a new release

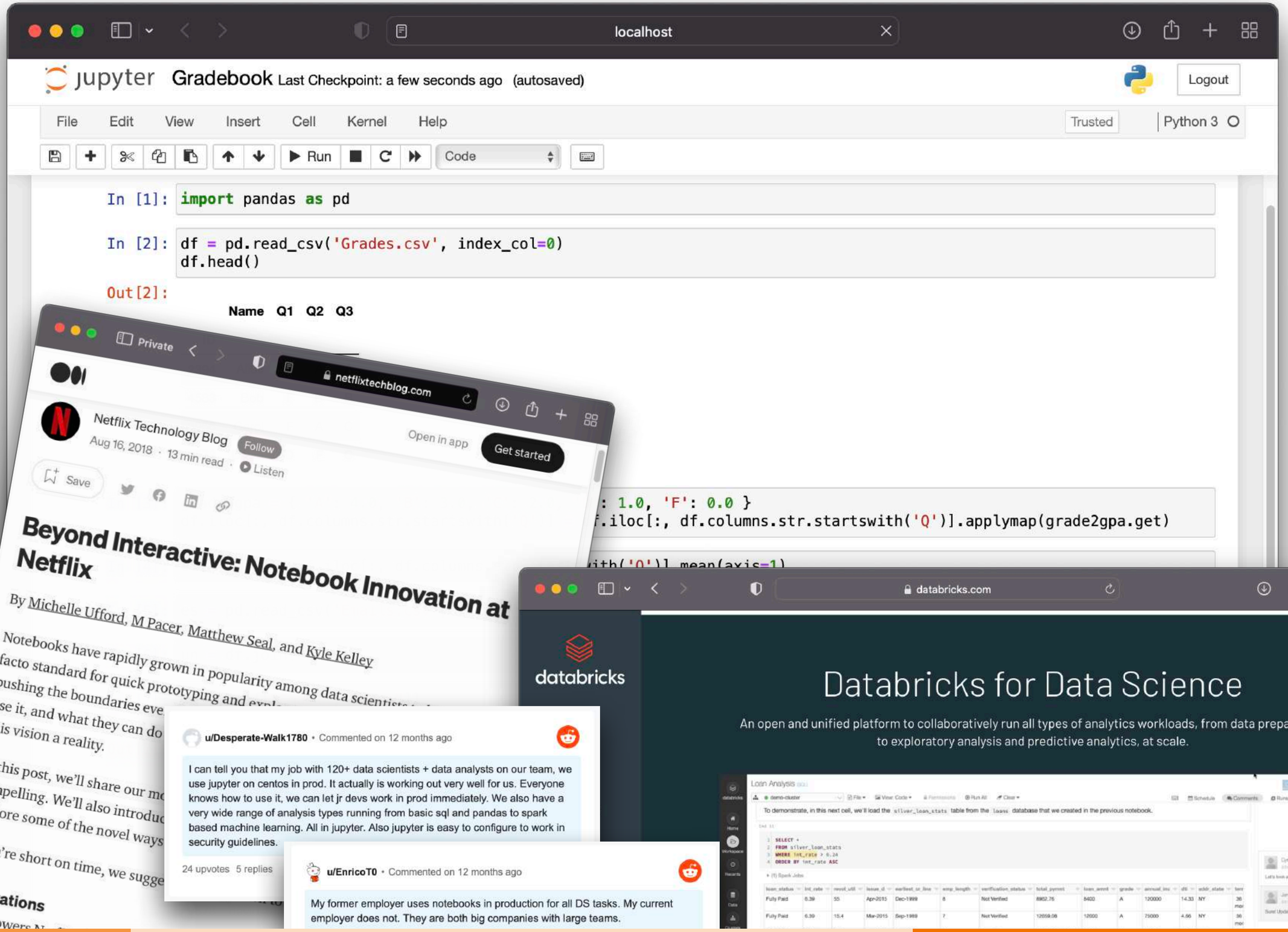
Packages

No packages published
Publish your first package

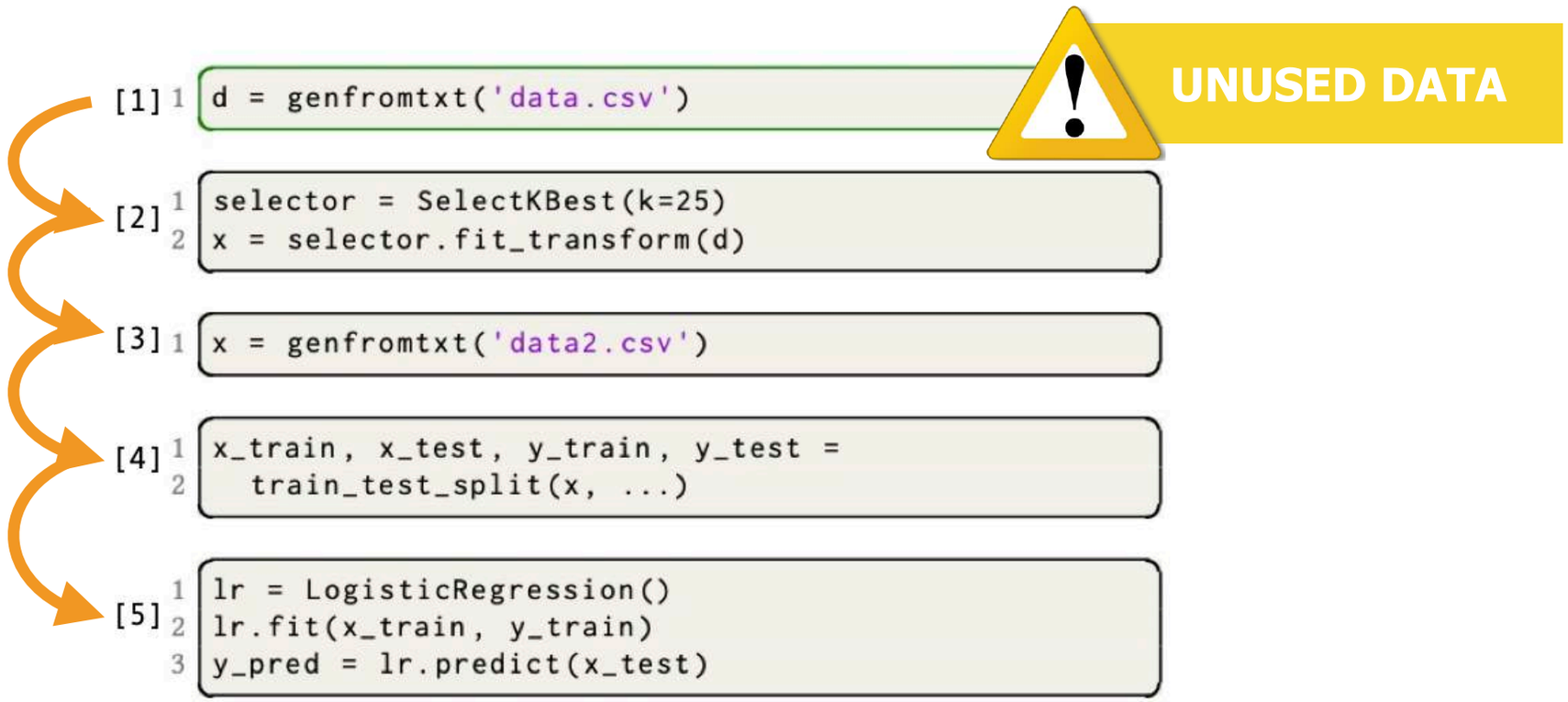
Contributors 9

Deployments 188

Jupyter Notebooks

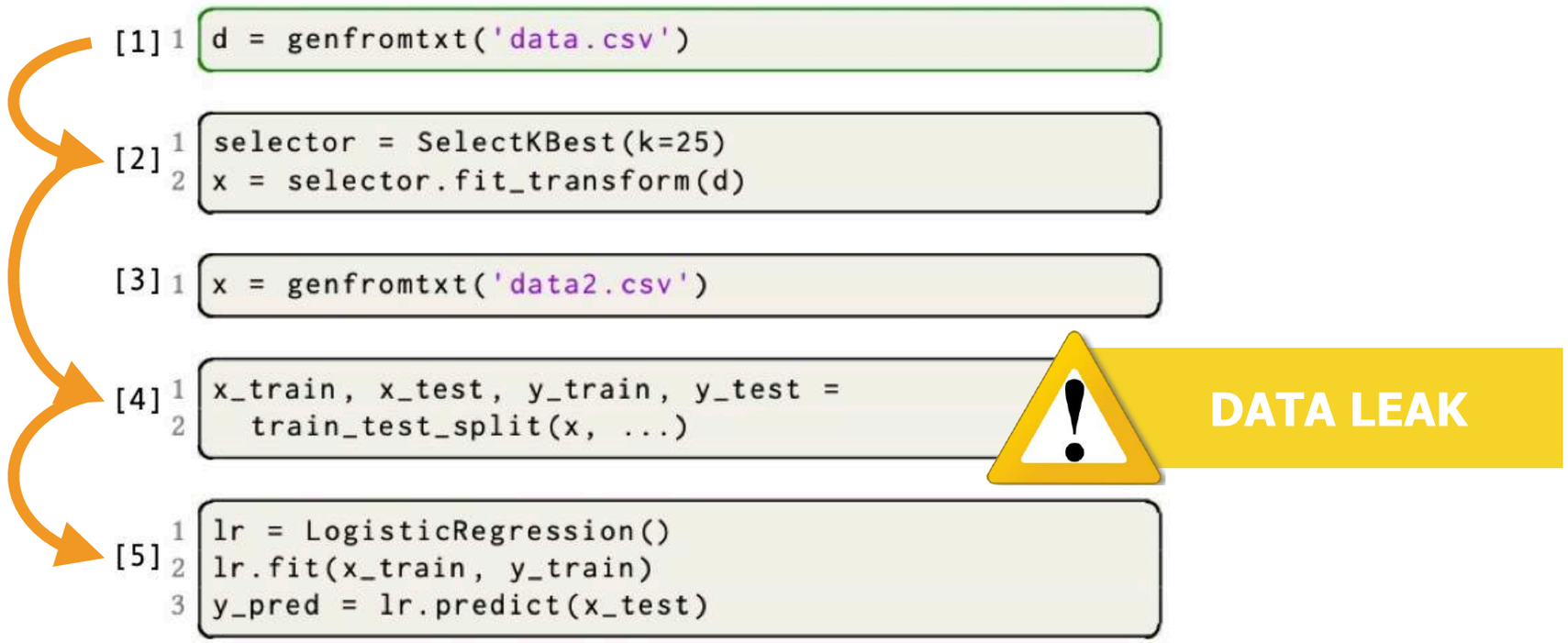


Jupyter Notebooks



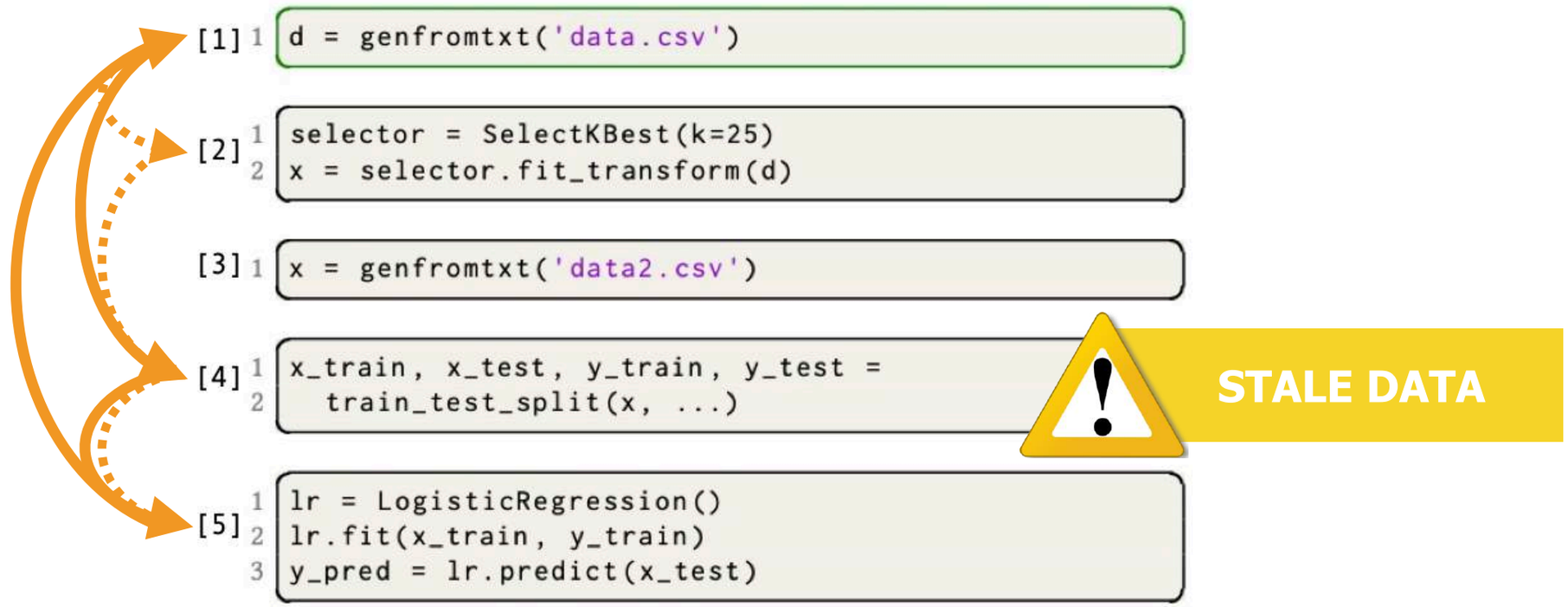
P. Subotić et al. - A Static Analysis Framework for Data Science Notebooks (ICSE 2022)

Jupyter Notebooks



P. Subotić et al. - A Static Analysis Framework for Data Science Notebooks (ICSE 2022)

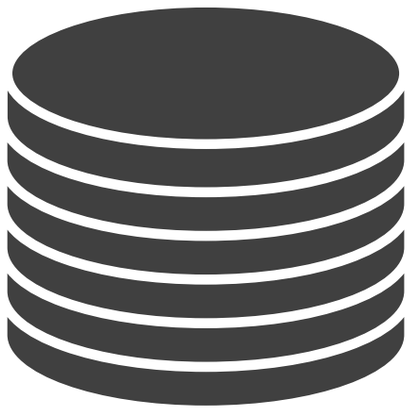
Jupyter Notebooks



P. Subotić et al. - A Static Analysis Framework for Data Science Notebooks (ICSE 2022)

Unexpected Data

Unexpected Data



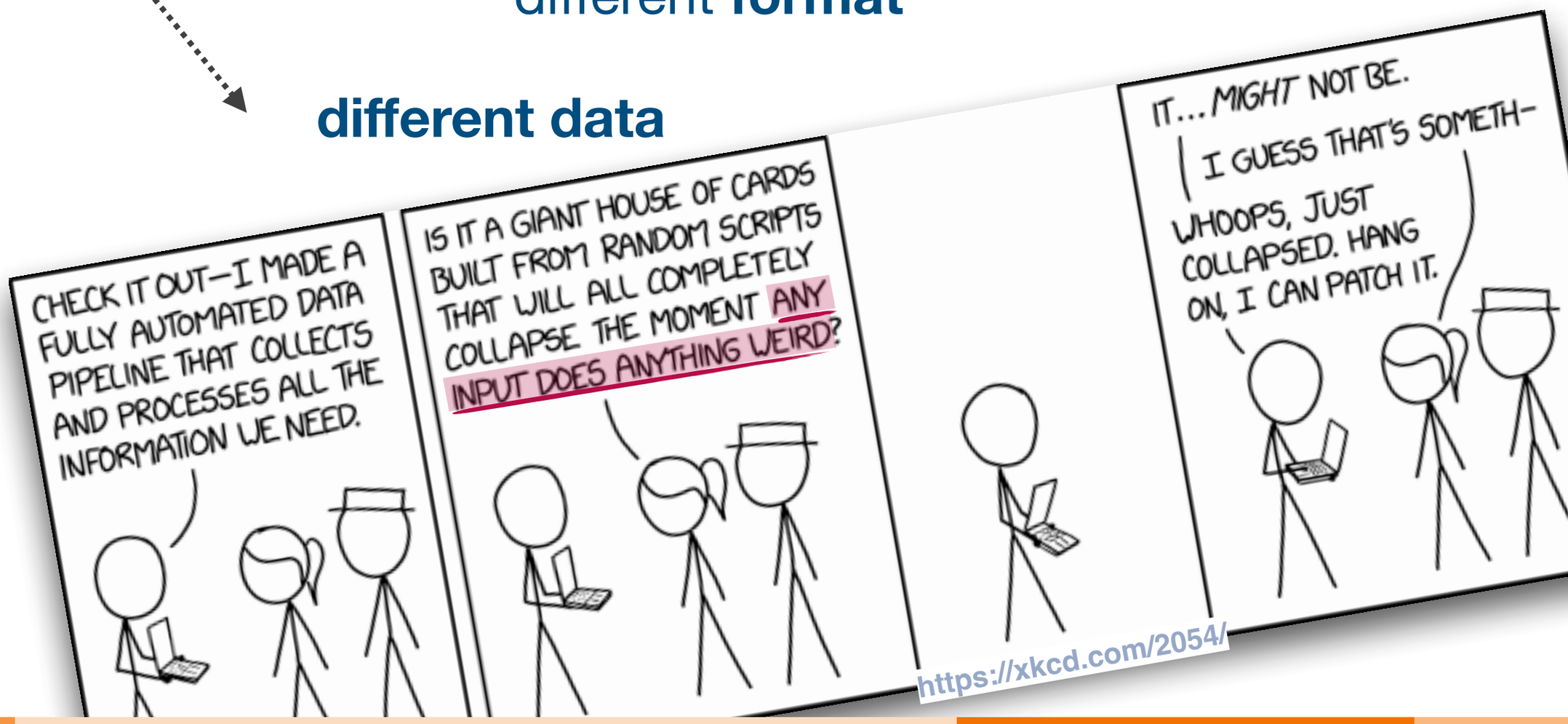
missing values

extra values

duplicate values

different **format**

different data



localhost

jupyter **Gradebook** Last Checkpoint: a few seconds ago (autosaved)

Logout

File Edit View Insert Cell Kernel Help Trusted Python 3

Run Code

In [1]:
```
import pandas as pd
```

In [2]:
```
df = pd.read_csv('Grades.csv', index_col=0)
df.head()
```

Out[2]:

ID	Name	Q1	Q2	Q3
2394	Alice	A	A	A
4583	Bob	F	B	B
3956	Carol	F	A	C
9578	David	D	F	C

In [3]:
```
grade2gpa = { 'A': 4.0, 'B': 3.0, 'C': 2.0, 'D': 1.0, 'F': 0.0 }
df.iloc[:, df.columns.str.startswith('Q')] = df.iloc[:, df.columns.str.startswith('Q')].applymap(grade2gpa.get)
```

In [4]:
```
df['Mean'] = df.iloc[:, df.columns.str.startswith('Q')].mean(axis=1)
```

In [5]:
```
es = pd.read_csv('Emails.csv', index_col=0)
```

In [6]:
```
un = df.join(es)
```

In [7]:
```
res = un[["Email", "Mean"]]
res.head()
```

Out[7]:

ID	Email	Mean
2394	alice@uni.eu	4.0
4583	bob@uni.eu	2.0
3956	carol@uni.eu	2.0
9578	david@uni.eu	1.0

Missing Values

jupyter Gradebook Last Checkpoint: a few seconds ago (autosaved)

File Edit View Insert Cell Kern

Run

```
In [1]: import pandas as pd

In [2]: df = pd.read_csv('Grad
        df.head()
```

Out[2]:

ID	Name	Q1	Q2	Q3
2394	Alice	A	A	A
4583	Bob	F	B	B
3956	Carol	F	A	C
9578	David	D	F	C

```
In [3]: grade2gpa = { 'A': 4.0
        df.iloc[:, df.columns.

In [4]: df['Mean'] = df.iloc[:

In [5]: es = pd.read_csv('Emai

In [6]: un = df.join(es)

In [7]: res = un[["Email", "Me
        res.head()
```

Out[7]:

ID	Email	Mean
2394	alice@uni.eu	4.0
4583	bob@uni.eu	2.0
3956	carol@uni.eu	2.0
9578	david@uni.eu	1.0

localhost

Logout

jupyter Gradebook Last Checkpoint: a minute ago (unsaved changes)

File Edit View Insert Cell Kernel Help

Trusted

Run Code

```
In [1]: import pandas as pd

In [2]: df = pd.read_csv('Grades.csv', index_col=0)
        df.head()
```

Out[2]:

ID	Name	Q1	Q2	Q3
2394	Alice	A	A	A
4583	Bob	F	B	B
3956	Carol	NaN	A	C
9578	David	D	F	C

```
In [3]: grade2gpa = { 'A': 4.0, 'B': 3.0, 'C': 2.0, 'D': 1.0, 'F': 0.0 }
        df.iloc[:, df.columns.str.startswith('Q')] = df.iloc[:, df.columns.str.startswith('Q')].applymap(grade2gp

In [4]: df['Mean'] = df.iloc[:, df.columns.str.startswith('Q')].mean(axis=1)

In [5]: es = pd.read_csv('Emails.csv', index_col=0)

In [6]: un = df.join(es)

In [7]: res = un[["Email", "Mean"]]
        res.head()
```

Out[7]:

ID	Email	Mean
2394	alice@uni.eu	4.0
4583	bob@uni.eu	2.0
3956	carol@uni.eu	3.0
9578	david@uni.eu	1.0

Extra Values

localhost

jupyter Gradebook Last Checkpoint: a few seconds ago (autosaved)

Logout

File Edit View Insert Cell Kern

Run

```
In [1]: import pandas as pd

In [2]: df = pd.read_csv('Grad
        df.head()
```

Out[2]:

ID	Name	Q1	Q2	Q3
2394	Alice	A	A	A
4583	Bob	F	B	B
3956	Carol	F	A	C
9578	David	D	F	C

```
In [3]: grade2gpa = { 'A': 4.0
        df.iloc[:, df.columns.

In [4]: df['Mean'] = df.iloc[:

In [5]: es = pd.read_csv('Emai

In [6]: un = df.join(es)

In [7]: res = un[["Email", "Me
        res.head()
```

Out[7]:

ID	Email	Mean
2394	alice@uni.eu	4.0
4583	bob@uni.eu	2.0
3956	carol@uni.eu	2.0
9578	david@uni.eu	1.0

localhost

jupyter Gradebook Last Checkpoint: 2 minutes ago (unsaved changes)

File Edit View Insert Cell Kernel Help

Trusted

Run Code

```
In [1]: import pandas as pd

In [2]: df = pd.read_csv('Grades.csv', index_col=0)
        df.head()
```

Out[2]:

ID	Name	Q1	Q2	Q3
2394	Alice	A	A	A	A
4583	Bob	F	B	B	NaN
3956	Carol	F	A	C	NaN
9578	David	D	F	C	NaN

```
In [3]: grade2gpa = { 'A': 4.0, 'B': 3.0, 'C': 2.0, 'D': 1.0, 'F': 0.0 }
        df.iloc[:, df.columns.str.startswith('Q')] = df.iloc[:, df.columns.str.startswith('Q')].applymap(grade2gp

In [4]: df['Mean'] = df.iloc[:, df.columns.str.startswith('Q')].mean(axis=1)

In [5]: es = pd.read_csv('Emails.csv', index_col=0)

In [6]: un = df.join(es)

In [7]: res = un[["Email", "Mean"]]
        res.head()
```

Out[7]:

ID	Email	Mean
2394	alice@uni.eu	4.0
4583	bob@uni.eu	3.0
3956	carol@uni.eu	3.0
9578	david@uni.eu	1.0

Different Formatted Values

jupyter Gradebook Last Checkpoint: a few seconds ago (autosaved)

Logout

File Edit View Insert Cell Kern

Run

```
In [1]: import pandas as pd

In [2]: df = pd.read_csv('Grad
        df.head()
```

Out[2]:

ID	Name	Q1	Q2	Q3
2394	Alice	A	A	A
4583	Bob	F	B	B
3956	Carol	F	A	C
9578	David	D	F	C

```
In [3]: grade2gpa = { 'A': 4.0
        df.iloc[:, df.columns.

In [4]: df['Mean'] = df.iloc[:

In [5]: es = pd.read_csv('Emai

In [6]: un = df.join(es)

In [7]: res = un[["Email", "Me
        res.head()
```

Out[7]:

ID	Email	Mean
2394	alice@uni.eu	4.0
4583	bob@uni.eu	2.0
3956	carol@uni.eu	2.0
9578	david@uni.eu	1.0

localhost

jupyter Gradebook Last Checkpoint: 14 minutes ago (unsaved changes)

File Edit View Insert Cell Kernel Help

Trusted

Run Code

```
In [1]: import pandas as pd

In [2]: df = pd.read_csv('Grades.csv', index_col=0)
        df.head()
```

Out[2]:

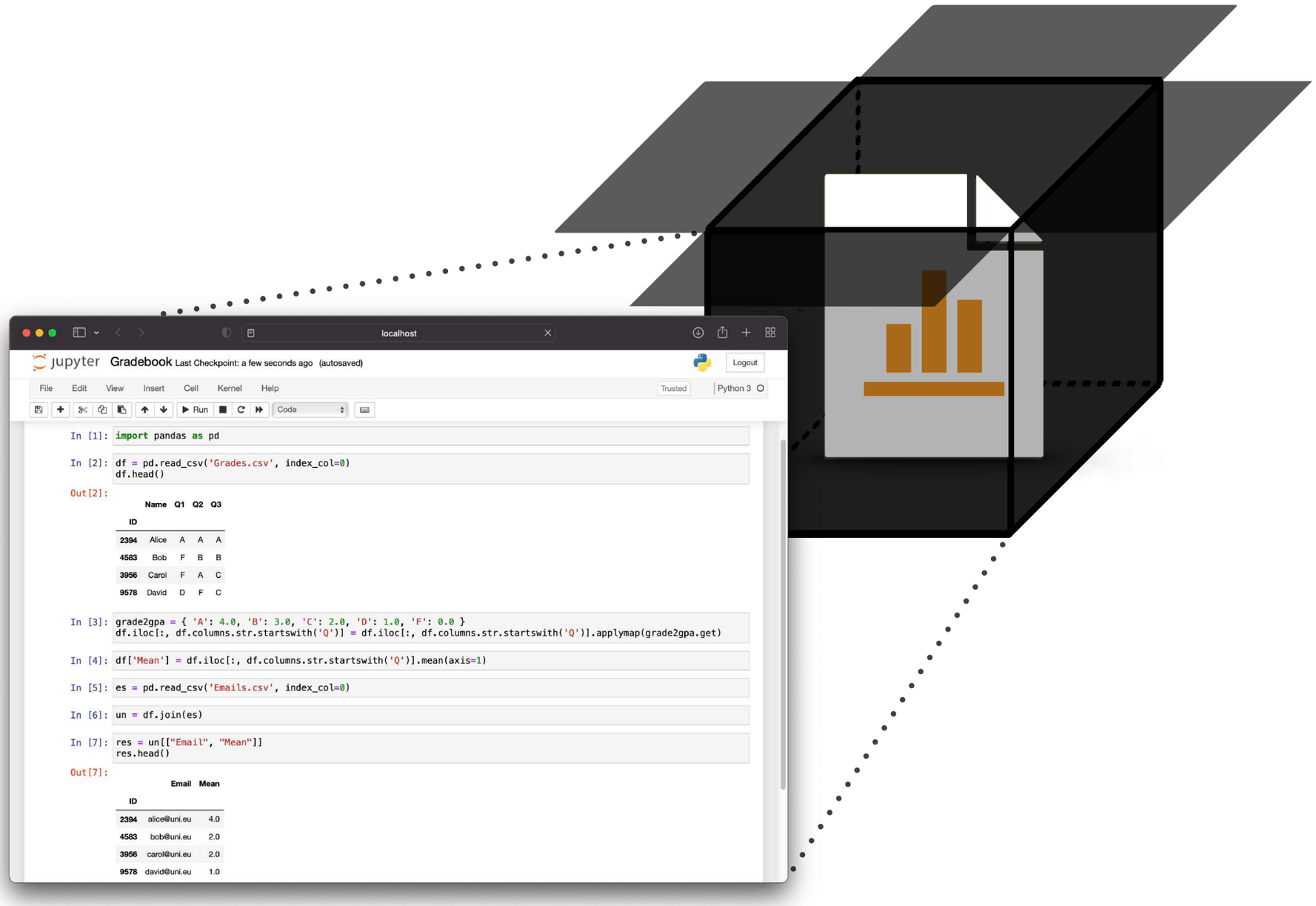
ID	Name	Q1	Q2	Q3
2394	Alice	A	A	A
4583	Bob	F	B+	B
3956	Carol	F	A	C
9578	David	D	F	C

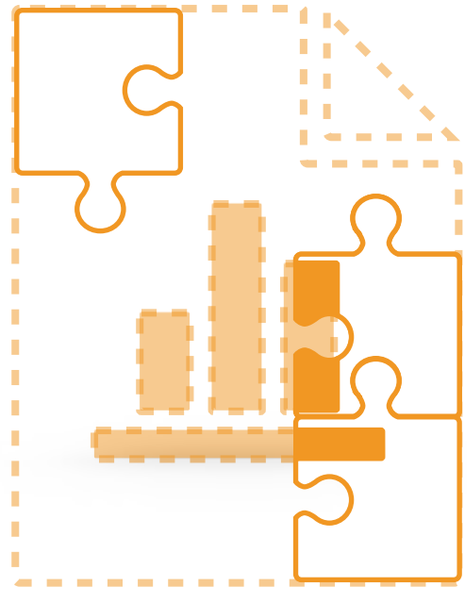
```
In [3]: grade2gpa = { 'A': 4.0, 'B': 3.0, 'C': 2.0, 'D': 1.0, 'F': 0.0 }
        df.iloc[:, df.columns.str.startswith('Q')] = df.iloc[:, df.columns.str.startswith('Q')].applymap(grade2gp

In [4]: df['Mean'] = df.iloc[:, df.columns.str.startswith('Q')].mean(axis=1)

In [5]: es = pd.read_csv('Emails.csv', index_col=0)

In [6]: un = df.join(es)

In [7]: res = un[["Email", "Mean"]]
        res.head()
```

Out[7]:

ID	Email	Mean
2394	alice@uni.eu	4.0
4583	bob@uni.eu	1.5
3956	carol@uni.eu	2.0
9578	david@uni.eu	1.0

Expected Data Analysis

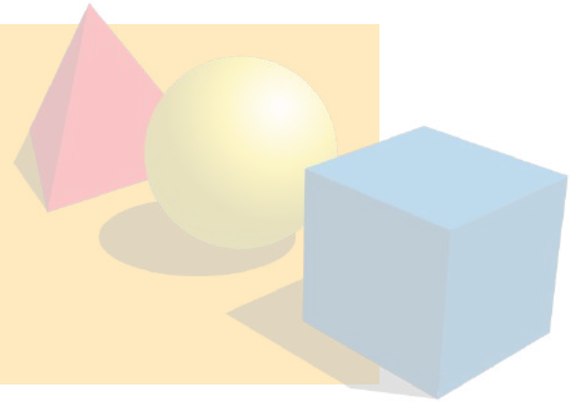




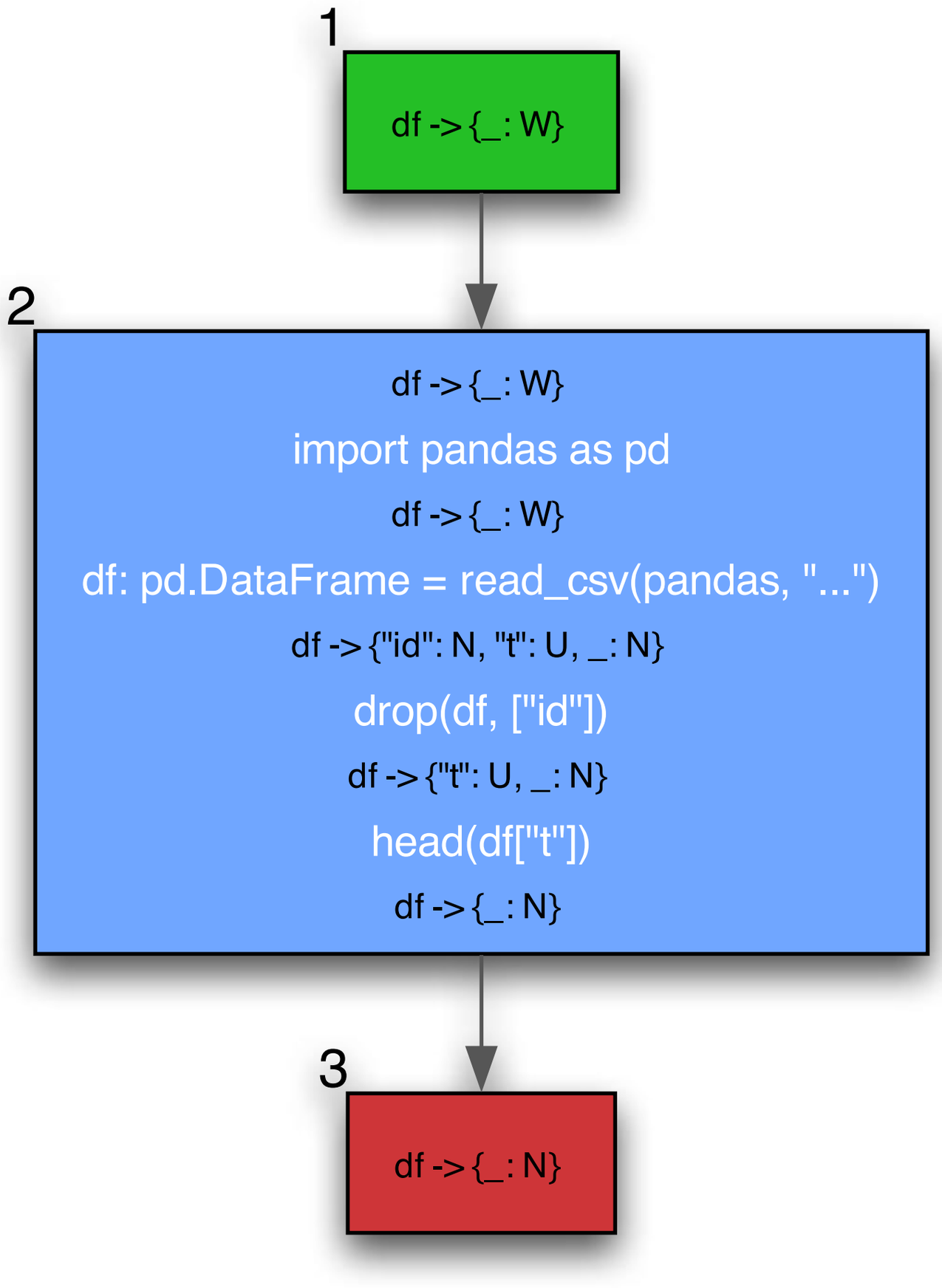
Expected Unused Data Analysis

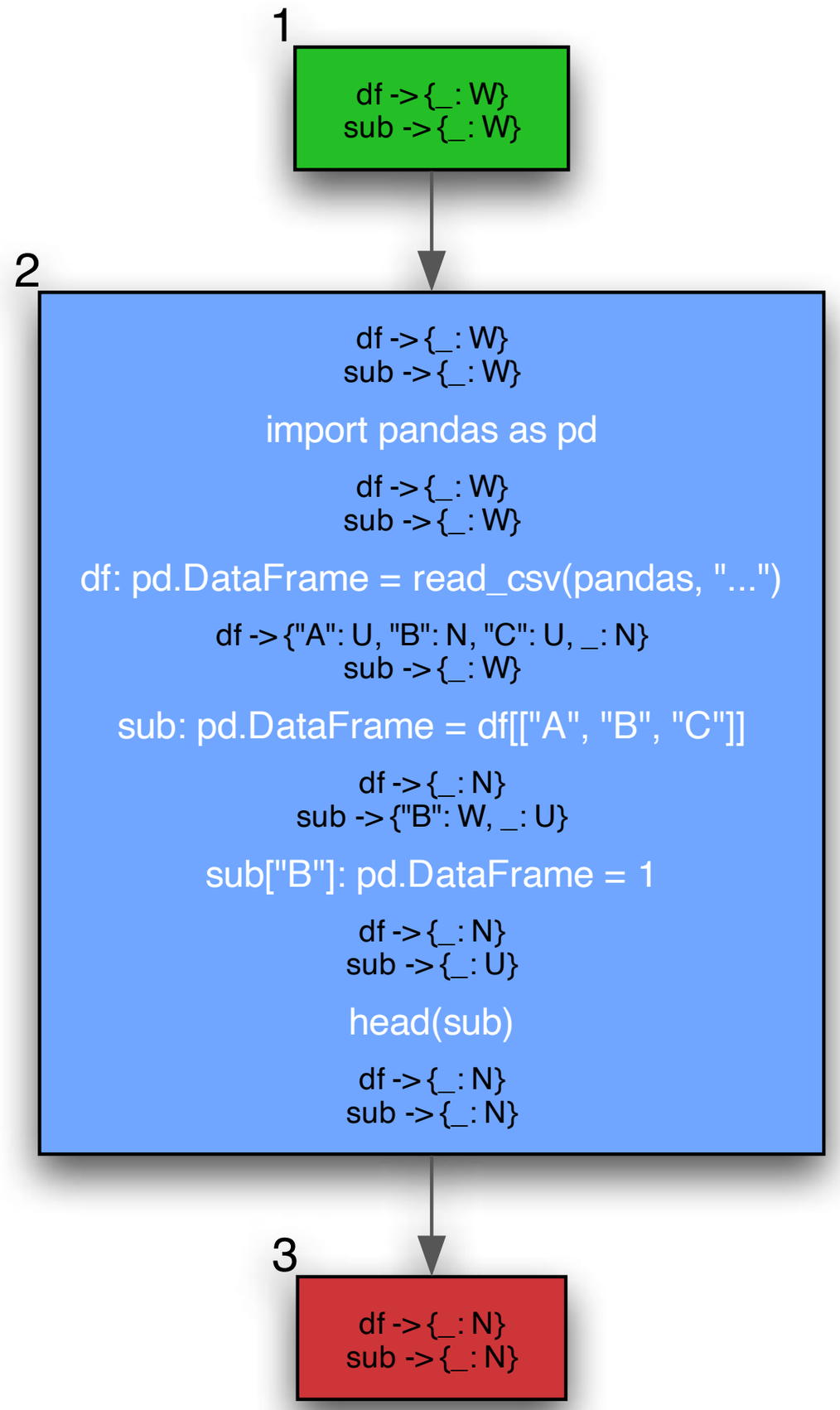
practical tools
targeting specific programs

algorithmic approaches
to decide program properties



mathematical models
of the program behavior


1
df -> {_: W}
2
df -> {_: W}
import pandas as pd
df -> {_: W}
df: pd.DataFrame = read_csv(pandas, "...")
df -> {"id": N, "t": U, _: N}
drop(df, ["id"])
df -> {"t": U, _: N}
head(df["t"])
df -> {_: N}
3
df -> {_: N}


1
df -> {_: W}
sub -> {_: W}
2
df -> {_: W}
sub -> {_: W}
import pandas as pd
df -> {_: W}
sub -> {_: W}
df: pd.DataFrame = read_csv(pandas, "...")
df -> {"A": U, "B": N, "C": U, _: N}
sub -> {_: W}
sub: pd.DataFrame = df[["A", "B", "C"]]
df -> {_: N}
sub -> {"B": W, _: U}
sub["B"]: pd.DataFrame = 1
df -> {_: N}
sub -> {_: U}
head(sub)
df -> {_: N}
sub -> {_: N}
3
df -> {_: N}
sub -> {_: N}

Bibliography

[Urban18] **Caterina Urban and Peter Müller**. An Abstract Interpretation Framework for Input Data Usage. In ESOP, 2018.

[Assaf17] **Mounir Assaf, David A. Naumann, Julien Signoles, Éric Totel, and Frédéric Tronel**. Hypercollecting Semantics and Its Application to Static Analysis of Information Flow. In POPL, 2017.

[Giegerich81] **Robert Giegerich, Ulrich Möncke, and Reinhard Wilhelm**. Invariance of Approximate Semantics with Respect to Program Transformations. 1981.

[Urban19] **Caterina Urban**. Static Analysis of Data Science Software. In SAS, 2019.

[Urban20] **Caterina Urban**. What Programs Want: Automatic Inference of Input Data Specifications. https://arxiv.org/abs/2007.10688, 2020.