Monitoring Data Minimization
And Other Similar Hyperproperties

Gerardo Schneider

Dept. of Computer Science and Engineering

Chalmers | University of Gothenburg

Sweden

CHALMERS

UNIVERSITY OF GOTHENBURG

Vetenskapsrådet

ARVI Winter School
France
19-21 March 2018

General Data Protection Regulation* (GDPR)

"Personal data must be: adequate, relevant, and limited to the minimum necessary in relation to the purposes for which they are processed…"

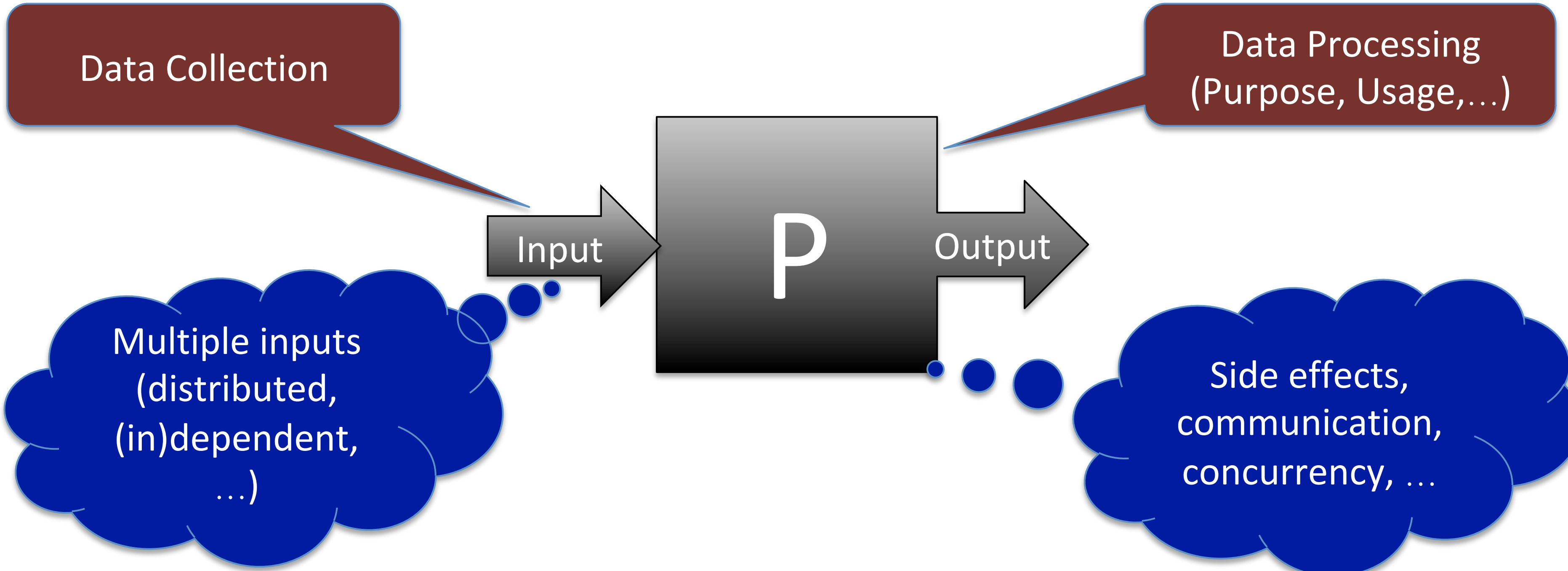
* EU – 2016/679: Will enter into application 25 May 2018

General Data Protection Regulation (GDPR)

"Personal data must be: **adequate**, **relevant**, and limited to **the minimum necessary** in relation to the **purposes** for which they are processed…"

- What might this mean?
- How can it be ensured?
  - Statically?
  - At runtime?

Scope



Collection vs Usage: We focus on the first (restricted)

This lecture:

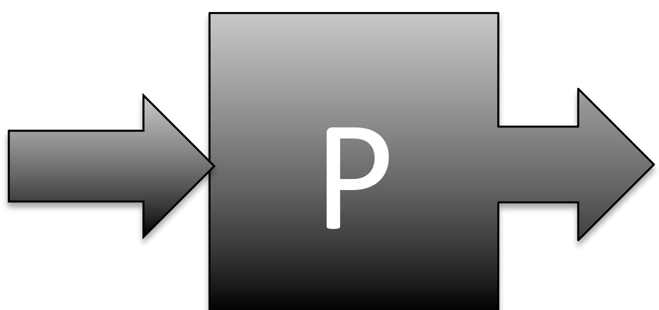
- **Part I**: Definitions + what can be done statically
- **Part II**: Monitoring data minimization, and extensions to other similar hyperproperties

PART I

On the concept of *data minimization* and what can be done statically

Minimality

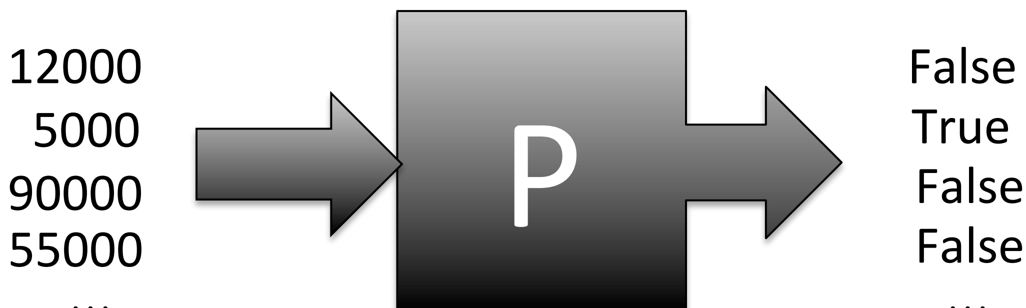
“Personal data must be: adequate, relevant, and limited to the **minimum necessary** in relation to the **purposes** for which they are processed…”



purpose = program

A Simple Program…

```
input(salary);
benefits := (salary < 10000);
output(benefits);
```



Is the information about the salary really needed?

Our Aim

- *Understand* and *define* data minimization from a programming language perspective
  - Minimality for *deterministic* programs (functions)

- *Determine* whether a given program is minimal or not

- *Compute* a data minimizer for a given program

It’s just Information Flow

Strong dependency [Cohen‘76]

Output *y* of P **strongly dependent** on input *x*
if some variation of *x* causes variation in *y*

Non-interference [Goguen & Meseguer’82]

Absence of strong dependency between input *secret (high)* and output *public (low)*

Non-interference

Non-interference concepts almost always talk about sufficient information for a given task:

- The *low* input is **sufficient** to compute the *low* output

Minimality needs to talk about **necessary** and **sufficient** information

Total Dependency

Strong dependency [Cohen'76]

Output *y* of P **strongly dependent** on input *x*
if ***some*** variation of *x* causes variation in *y*

Total dependency

Output *y* of P **totally dependent** on input *x*
if **any** variation of *x* causes variation in *y*

Data Minimality

Definition

P is data minimal if its output is ***totally dependent*** on its inputs

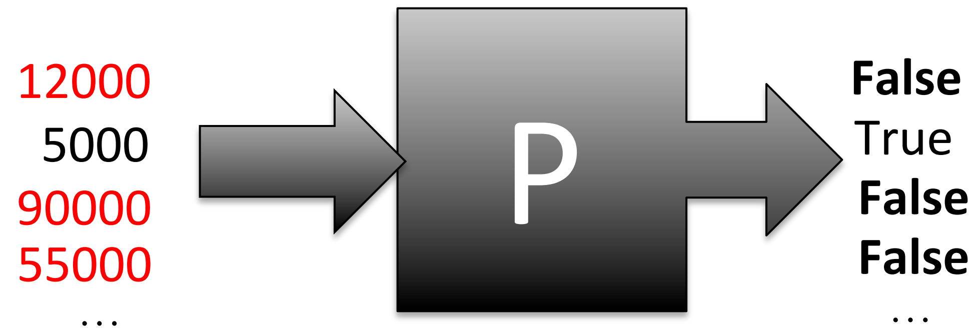
Two variants:

- **Monolithic**: a single input source
- **Distributed**: multiple independent sources

Monolithic case: minimality is just *injectivity*

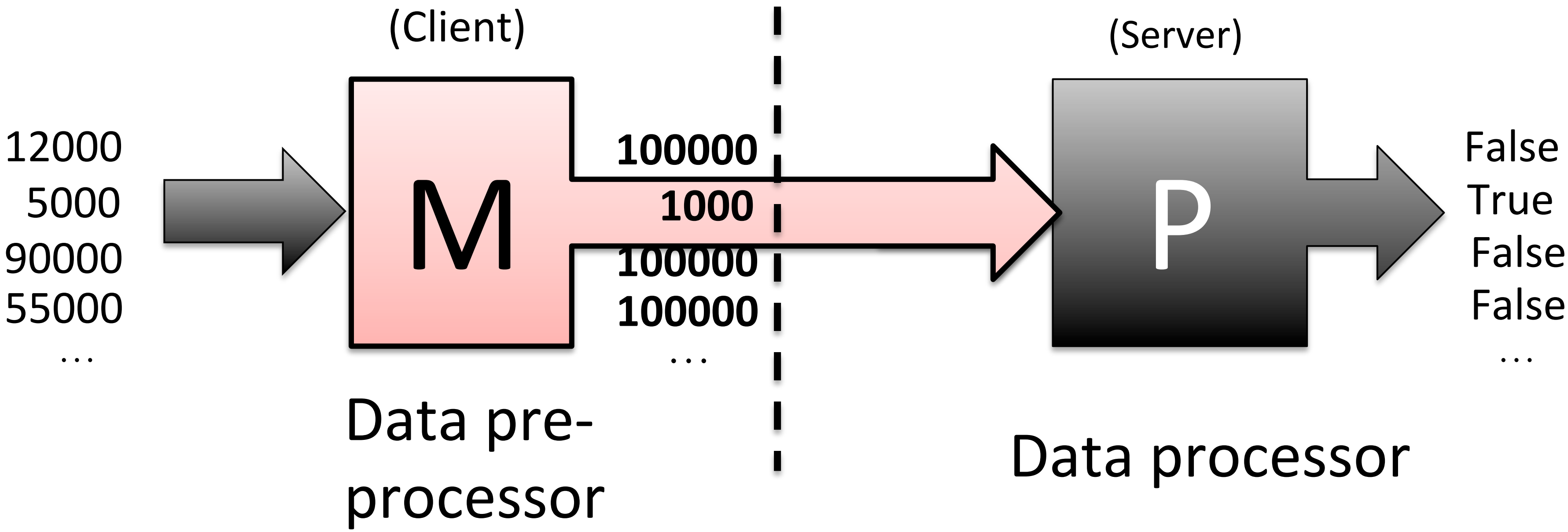
Minimization

P (the "benefits" program) is not minimal



Minimization

M can replace actual values with canonical representative values which are good enough



The minimizer M:

$$\text{salary} \mapsto \begin{cases} 1000 & \text{if salary} < 10000 \\ 100000 & \text{otherwise} \end{cases}$$

Pre-Processors

Definition

M is a *monolithic pre-processor* for P iff

$$P \circ M = P$$

$$M \circ M = M$$

M is a *monolithic minimizer* if $P|_{\mathrm{range}(M)}$ is minimal

Key Properties of Minimizers

Kernel of a function:

$$\ker(F) = \{ (x,y) \mid F(x) = F(y) \}$$

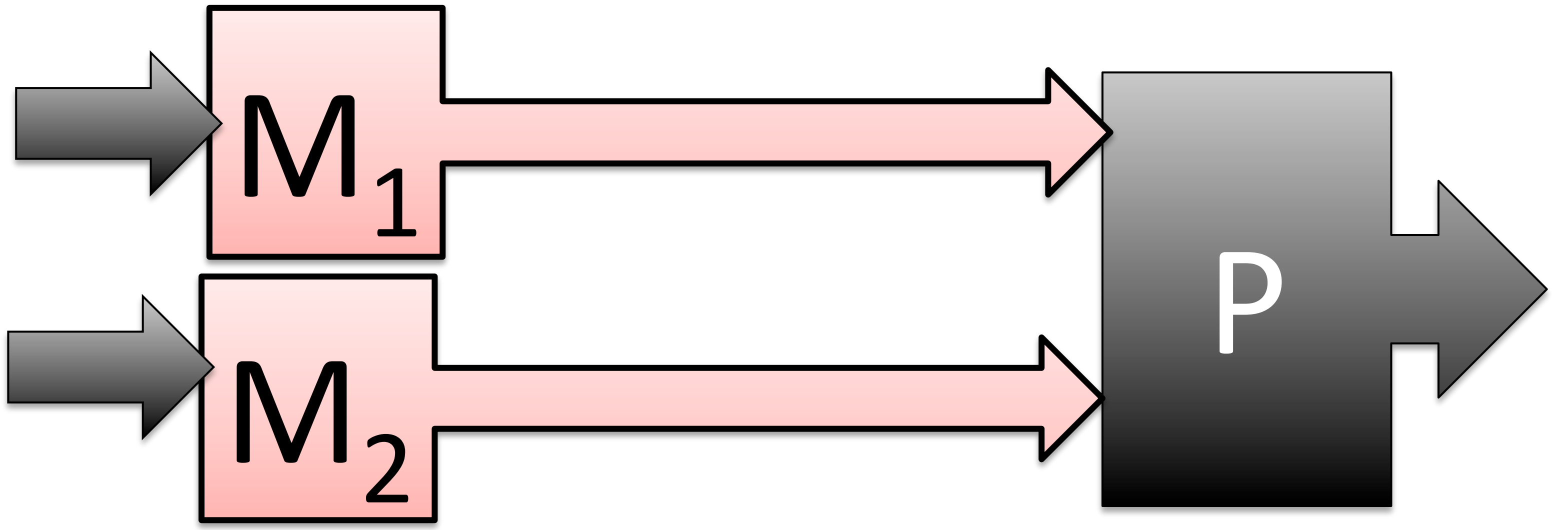
(Induces a partition of the input s.t. each input in the same equivalence class is mapped to the same output)

- M is a **minimizer** iff ker(M) = ker(P)

Theorem:

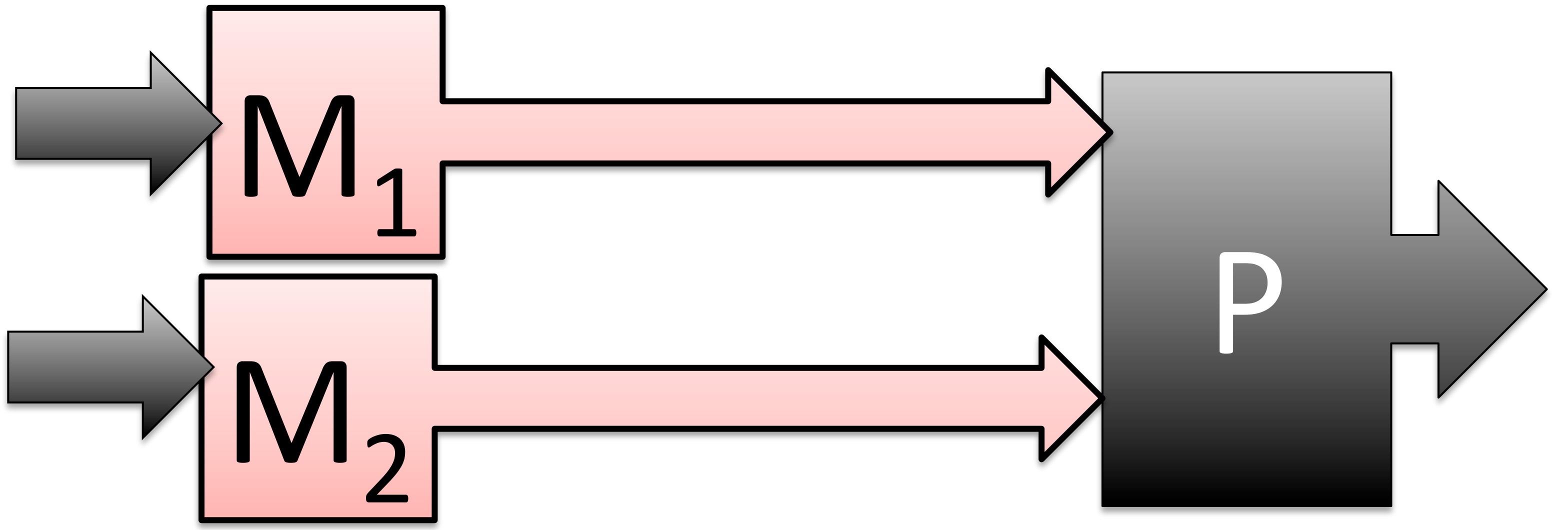
For every P there exists a monolithic minimizer

Distributed Minimizer



But what's the definition when P receives inputs from different sources?

Distributed Minimizer

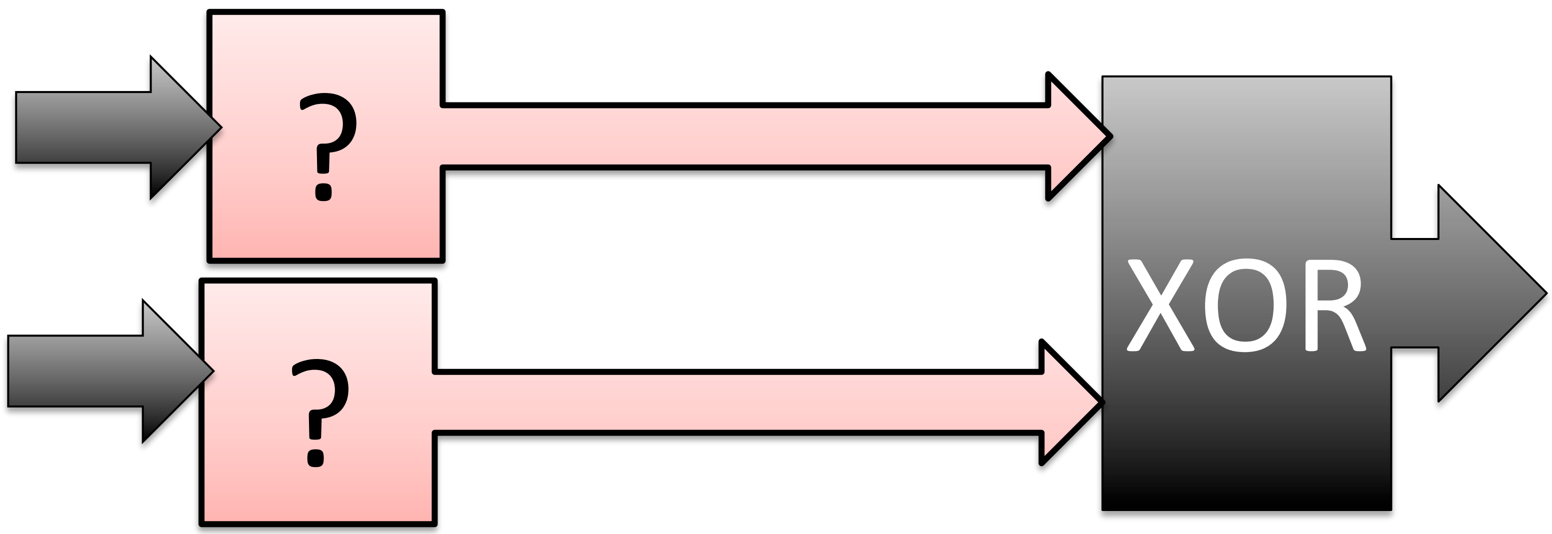


Intuition: "distributed minimizer" is the best we can do when each input is processed independently

Definition

For all input vectors **u** and **v** differing in just one position, P(**u**) and P(**v**) are different

Distributed Minimizer



Intuition: “distributed minimizer” is the best we can do when each input is processed independently

Definition

For all input vectors **u** and **v** differing in just one position, P(**u**) and P(**v**) are different

Distributed Minimality

XOR (F,T) = T

XOR (T,F) = T

XOR (F,F) = F

XOR (T,T) = F

Distributed minimal

XOR(**F**,**F**) ≠ XOR(**T**,**F**),
XOR(**T**,**T**) ≠ XOR(**F**,**T**),
XOR(**F**,**F**) ≠ XOR(**F**,**T**),
XOR(**T**,**T**) ≠ XOR(**T**,**F**)

But **not** monolithic minimal

XOR(F,F) = XOR(T,T)
but
(F,F) ≠ (T,T)

Necessarily *weaker* than monolithic minimality

Distributed Minimality

Too strong a notion!

OR (F,F) = F

OR (F,T) = T

OR (T,F) = T

OR (T,T) = T

Not distributed minimal

OR(**F**,**T**) **=** OR(**T**,**T**)

Not possible to minimize either argument further independently of the other

Distributed Minimality

Definition (first definition)

P is strongly distributed minimal iff

For all input vectors **u** and **v** differing in just one position, P(**u**) and P(**v**) are different

Definition

P is distributed minimal iff

For any input position i and distinct values x and y

there exists **u** such that $P(\mathbf{u}[i \mapsto x]) \neq P(\mathbf{u}[i \mapsto y])$

Distributed Minimality

OR (F,F) = F

OR (F,T) = T

OR (T,F) = T

OR (T,T) = T

Distributed minimal

For (**F**,***x***) and (**T**,***x***) we can always find a value for ***x*** s.t.
$OR(F,x) \neq OR(T,x)$ (e.g. ***x*** = **F**)

(Similarly for (*x*, **F**) and (*x*, **T**))

Properties of Distributed Minimizers

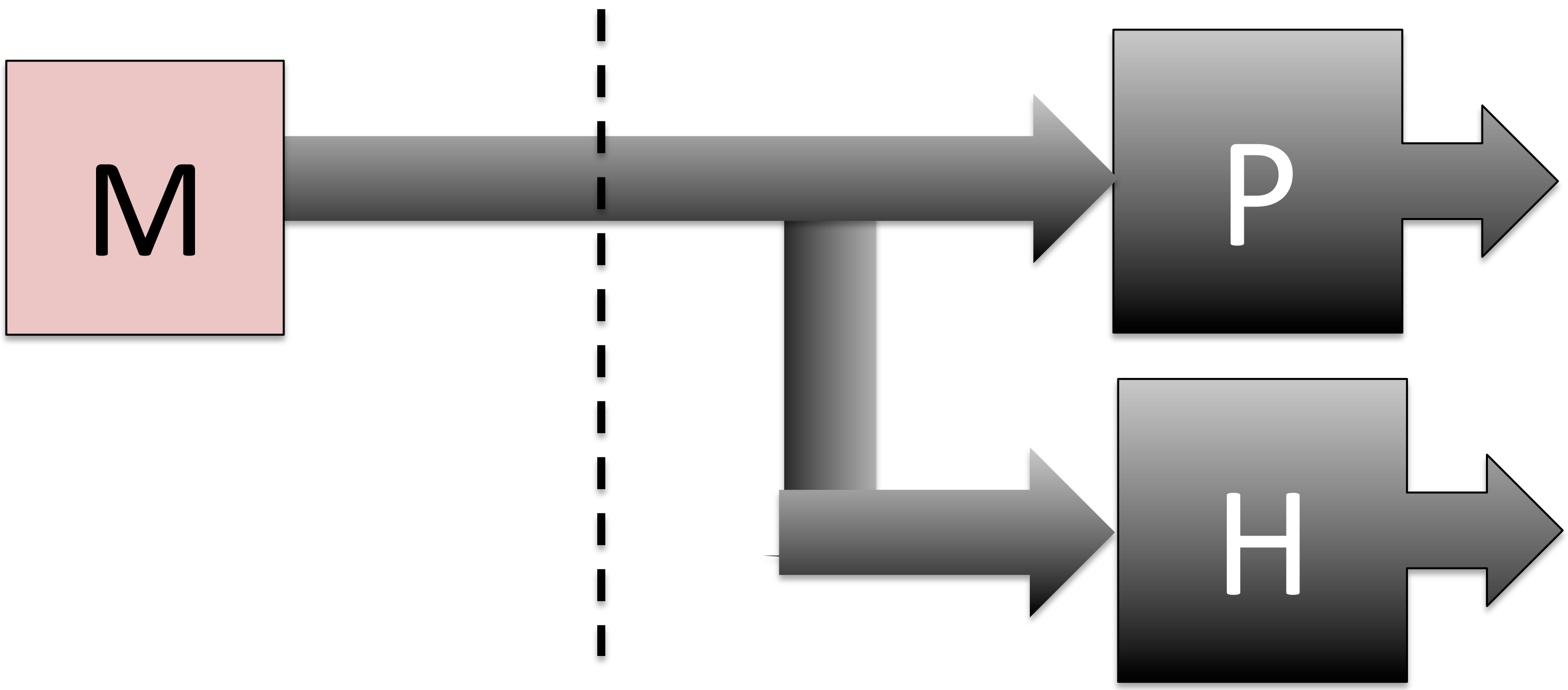
Theorem:

For every P there exists a distributed minimizer

Monolithic => Strong distributed => Distributed

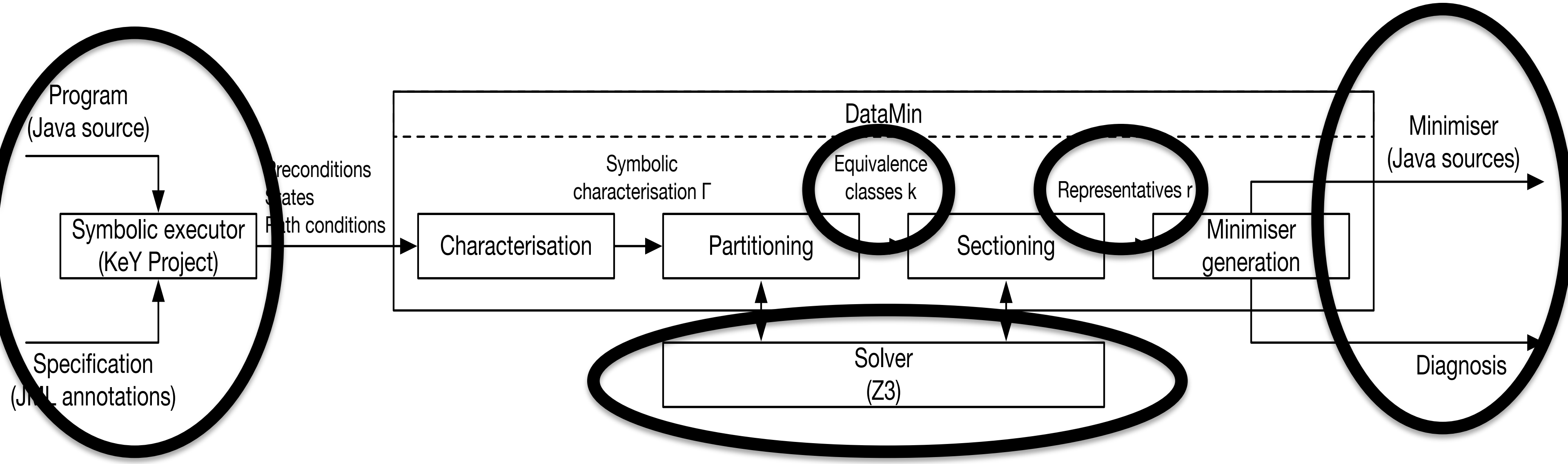
The Security Perspective

Attacker model: Hidden purpose ("secondary use")



Theorem: If M is the best minimizer, for all hidden uses H we have <P,H> o M = P

DataMin
Building Minimizers



Applied to

- Loyalty status (monolithic)
- Credit score (distributed)

Limitations… / Further Work

- Programs as functions
  - How to handle reactive/interactive programs?
- Distributed case not fine-grained enough
  - Partial knowledge between different sources, (inter)dependencies between sources
- Practicality: undecidability of static computation of the minimizer (automatically)
  - Even if computable, it's costly: enumeration of input and output domains

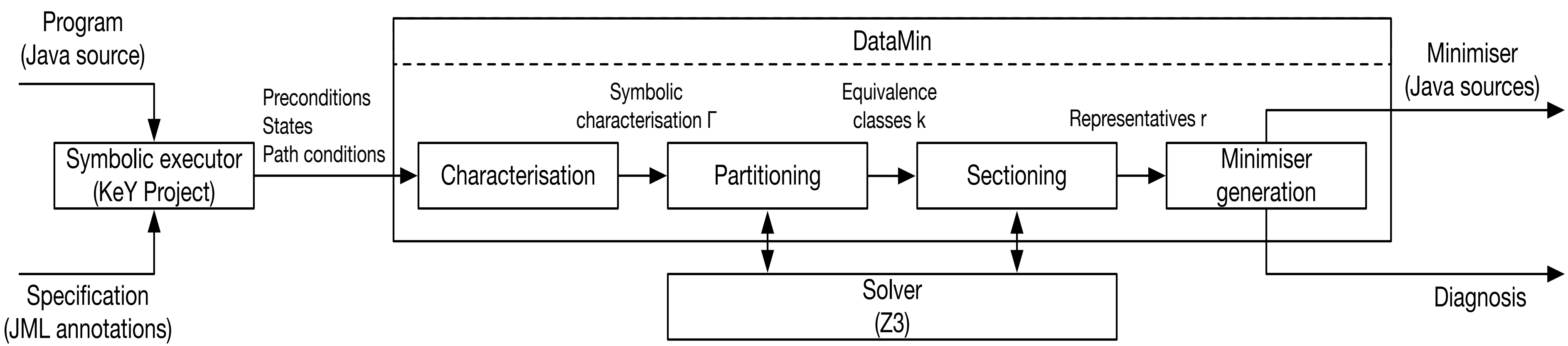
Summary

- Defined data minimization as a variant of strong dependency (*total* dependency)
  - For *deterministic* programs (functions)
- Provided criteria to determine whether a program is minimal or not
- Gave a semi-decision procedure (and proof-of-concept implementation) to **statically** compute a data minimizer for a given program
  - Based on symbolic execution and SAT solvers

PART II

Runtime monitoring /enforcement of *data minimization* and other similar *hyperproperties*

Why Monitoring?



Semi-decision procedure

Needs the source code

Cannot be applied if you want to check a third-party…

Can we do it?

- M.R. Clarkson, B. Finkbeiner, M. Koleini, K.K. Micinski, M.N. Rabe, C. Sánchez: *Temporal Logics for Hyperproperties*. POST'14
- B. Bonakdarpour and B. Finkbeiner. 2016. *Runtime Verification for HyperLTL.* In RV'16.
- S. Agrawal and B. Bonakdarpour. 2016. *Runtime Verification of k-Safety Hyperproperties in HyperLTL.* In CSF'16.
- N. Brett, U. Siddique, and B. Bonakdarpour. 2017. *Rewriting-Based Runtime Verification for Alternation-Free HyperLTL.* In TACAS'17.

Monitoring algorithms for the **alternation-free** fragment of **HyperLTL**

Universal quantifiers: usually **not** monitorable

Existential quantifiers: monitorable

What Does it Means for Us?

Data minimization may be expressed in HyperLTL!

We are done then! Somebody else did it!

Yes, but… Algorithms are general for HyperLTL

(Monolithic) Data Minimization Revisited…

Quantification over traces

$$\forall \pi, \forall \pi' : \pi_I \neq \pi'_I \implies \pi_O \neq \pi'_O.$$

HyperLTL$_{2S}$

$$\forall \pi, \forall \pi' : \psi$$

$$\psi ::= a_\pi \mid \neg\psi \mid \psi \vee \psi$$

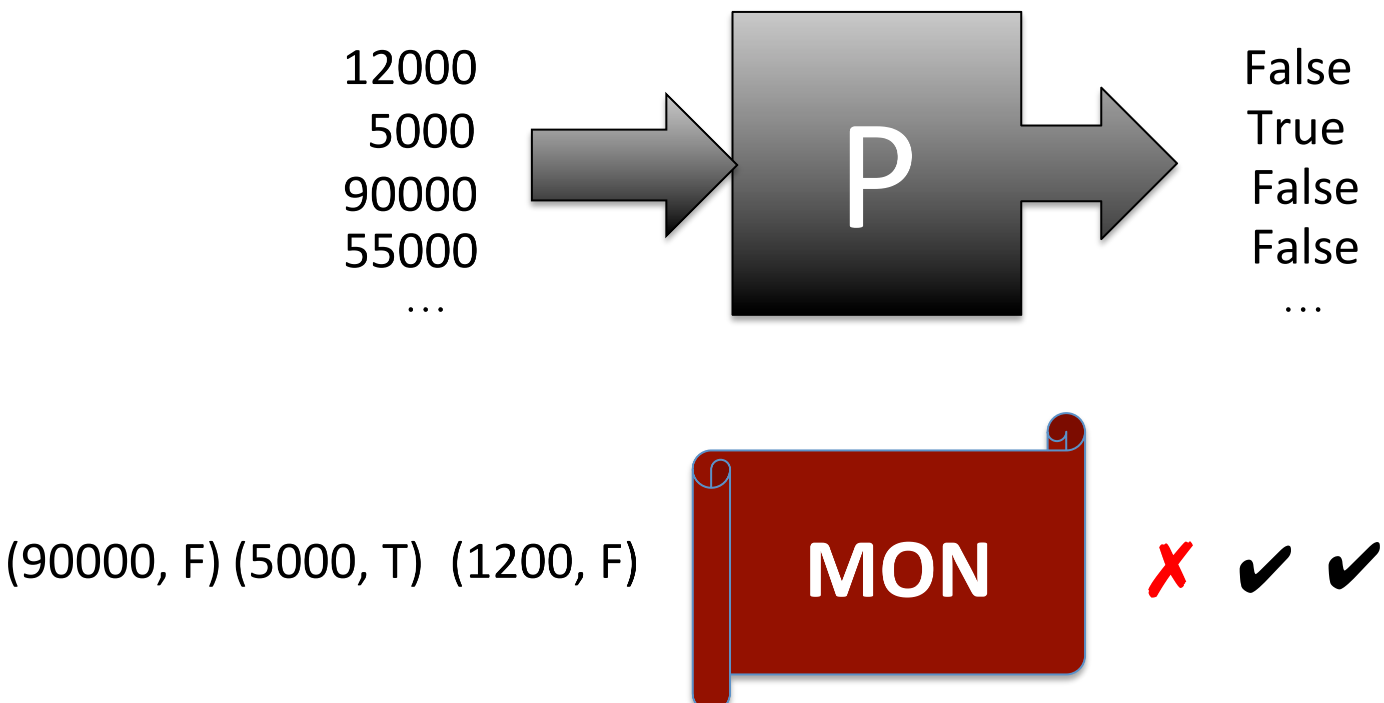
- ***Non-minimality*** *is* monitorable but it is in general impossible to give a final verdict for minimality!
- Traces are of fixed length (one)
- We are considering deterministic programs
- The property only talks about inputs and outputs!

It should be simpler to monitor!

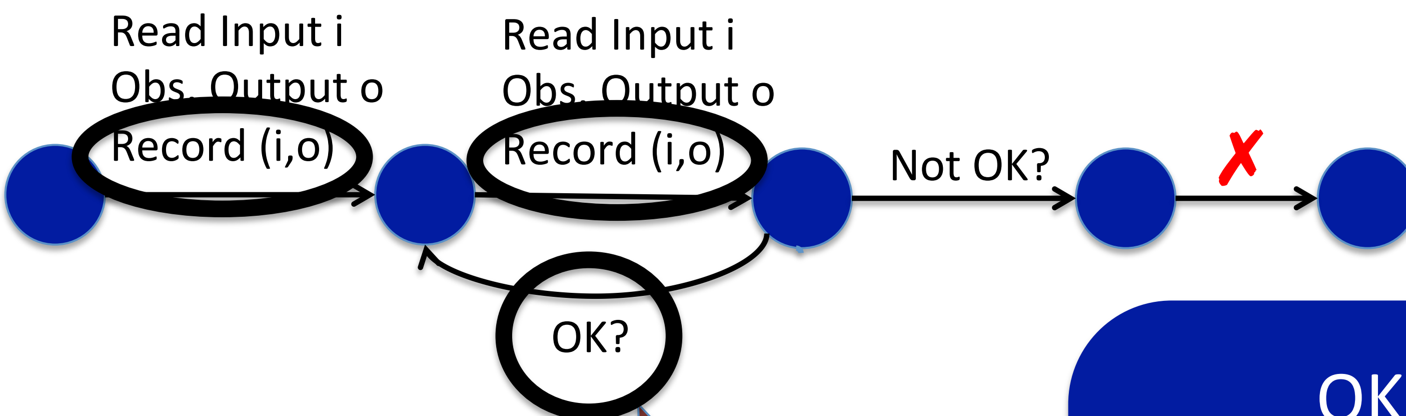
Monitoring Data Minimization (Program-in-loop)

Reduction to trace property

Not very important (algorithm is simpler)



Monitor for Data Minimization



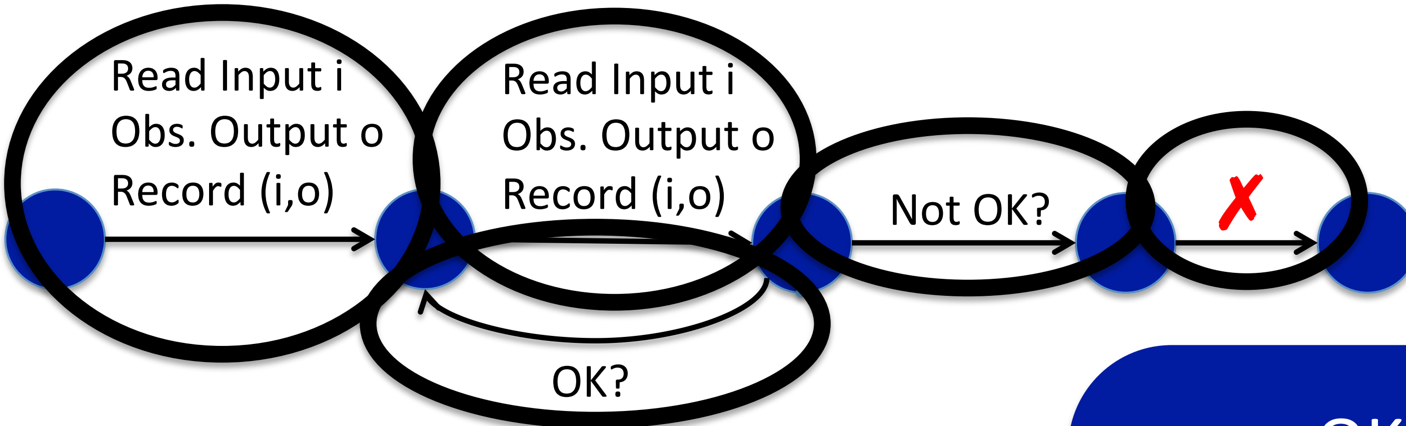
The “intelligence” is in the **OK?** predicate

Input	Output
i_1	o_1
i_2	o_2
i_3	o_3
…	…

OK?
(**Monolithic** Data Minimization)

Is there a prefix with the same output and different input?

Monitor for ***Monolithic*** Data Minimization



Input	Output
1200	F
5000	T
90000	T

OK?
(**Monolithic** Data Minimization)

Is there a prefix with the same output and different input?

Monitor for ***Distributed*** Data Minimization

- Monitor very similar to monolithic case but reading inputs from all sources independently
  - More states to read from all sources
- The **OK?** predicate will be different

- (A bit more complex − more theoretical results in Tech Rep *Monitoring Data Minimisation*)

Is that All?

Other properties *similar* to Data Minimization

- *Non-interference*
- *Integrity*
- Software doping (or rather *doping-free programs*)

Non-interference

- *P* satisfies non-interference if every pair of traces with the same (initial) *low* observation remains indistinguishable for low users

- Absence of strong dependency between input *secret (high)* and output *public (low)*
  - The (public) output observed by the low security users should only depend on low input information

$$\forall \pi, \forall \pi' : (\pi_{I,L} = \pi'_{I,L}) \implies (\pi_{O,L} = \pi'_{O,L})$$

Integrity

- Integrity requires that *high* behaviour of a system should not be influenced by *low* inputs (that can be potentially altered by a malicious user)

- Traces having the same high inputs but possibly different low inputs should have the same high outputs

$$\forall \pi, \forall \pi' : (\pi_{I,H} = \pi'_{I,H}) \implies (\pi_{O,H} = \pi'_{O,H})$$

Doping-Free Programs

- *P* is doping-free if small variations in the input produces small variations in the output

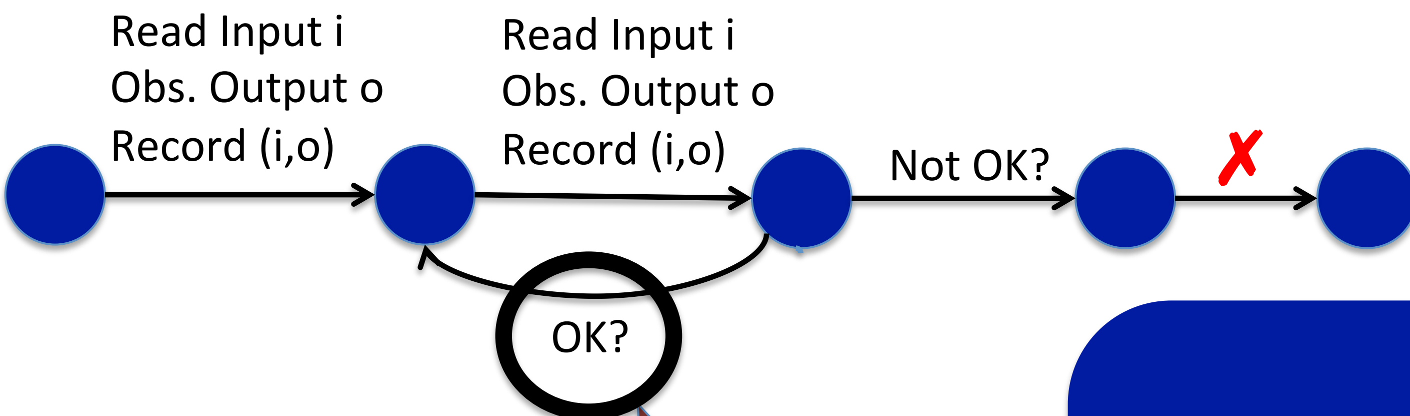
- A *parameterized* program *P* is doping-free if for all pairs of parameters of interest *p* and *p′*, and input *i*, then $P_p(i) = P_{p'}(i)$

$$\forall \pi, \forall \pi' : ((\pi_{Parm} \in PIntrs) \wedge (\pi'_{Parm} \in PIntrs) \wedge (\pi_I = \pi'_I)) \implies (\pi_O = \pi'_O)$$

Other properties *similar* to Data Minimization

Property	Property expressed in Hyper_{2S}
Data minimisation (Monolithic minimality)	$\forall\pi, \forall\pi' : \pi_I \neq \pi'_I \implies \pi_O \neq \pi'_O.$
Non-Interference	$\forall\pi, \forall\pi' : (\pi_{I,L} = \pi'_{I,L}) \implies (\pi_{O,L} = \pi'_{O,L})$
Integrity	$\forall\pi, \forall\pi' : (\pi_{I,H} = \pi'_{I,H}) \implies (\pi_{O,H} = \pi'_{O,H})$
Software doping (doping free program)	$\forall\pi, \forall\pi' : ((\pi_{Parm} \in PIntrs) \land (\pi'_{Parm} \in PIntrs) \land (\pi_I = \pi'_I)) \implies (\pi_O = \pi'_O)$
Strong distributed minimality	$\forall\pi, \forall\pi',$ let $\pi_I = (i_1, \cdots, i_n), \pi'_I = (i'_1, \cdots, i'_n).$ $(\exists x \in [1, n] : i_x \neq i'_x \land \forall y \in [1, n] : y \neq x \implies i_y = i'_y) \implies \pi_o \neq \pi'_o.$

Monitor for other Properties in HyperLTL$_{2S}$



Input	Output
i_1	o_1
i_2	o_2
i_3	o_3
…	…

The “intelligence” is in the **OK?** predicate

OK?

• • •

Parameterized Monitor for Properties in HyperLTL$_{2S}$

Property	Property expressed in Hyper$_{2S}$
Data minimisation (Monolithic minimality)	$\forall\pi, \forall\pi' : \pi_I \neq \pi'_I \implies \pi_O \neq \pi'_O$.
Non-Interference	$\forall\pi, \forall\pi' : (\pi_{I,L} = \pi'_{I,L}) \implies (\pi_{O,L} = \pi'_{O,L})$
Integrity	$\forall\pi, \forall\pi' : (\pi_{I,H} = \pi'_{I,H}) \implies (\pi_{O,H} = \pi'_{O,H})$
Software doping (doping free program)	$\forall\pi, \forall\pi'$: $((\pi_{Parm} \in PIntrs) \land (\pi'_{Parm} \in PIntrs) \land (\pi_I = \pi'_I)) \implies (\pi_O = \pi'_O)$
Strong distributed minimality	$\forall\pi, \forall\pi'$, let $\pi_I = (i_1, \cdots, i_n), \pi'_I = (i'_1, \cdots, i'_n)$. $(\exists x \in [1,n] : i_x \neq i'_x \land \forall y \in [1,n] : y \neq x \implies i_y = i'_y) \implies \pi_o \neq \pi'_o$.

OK?

. . .

Runtime Verification (Monitoring) but…

(an unimportant clarifying note)

- You have probably noted that the RV technique we are using here doesn't follow the "standard" way of getting the monitor (cf. e.g. Martin's talk)

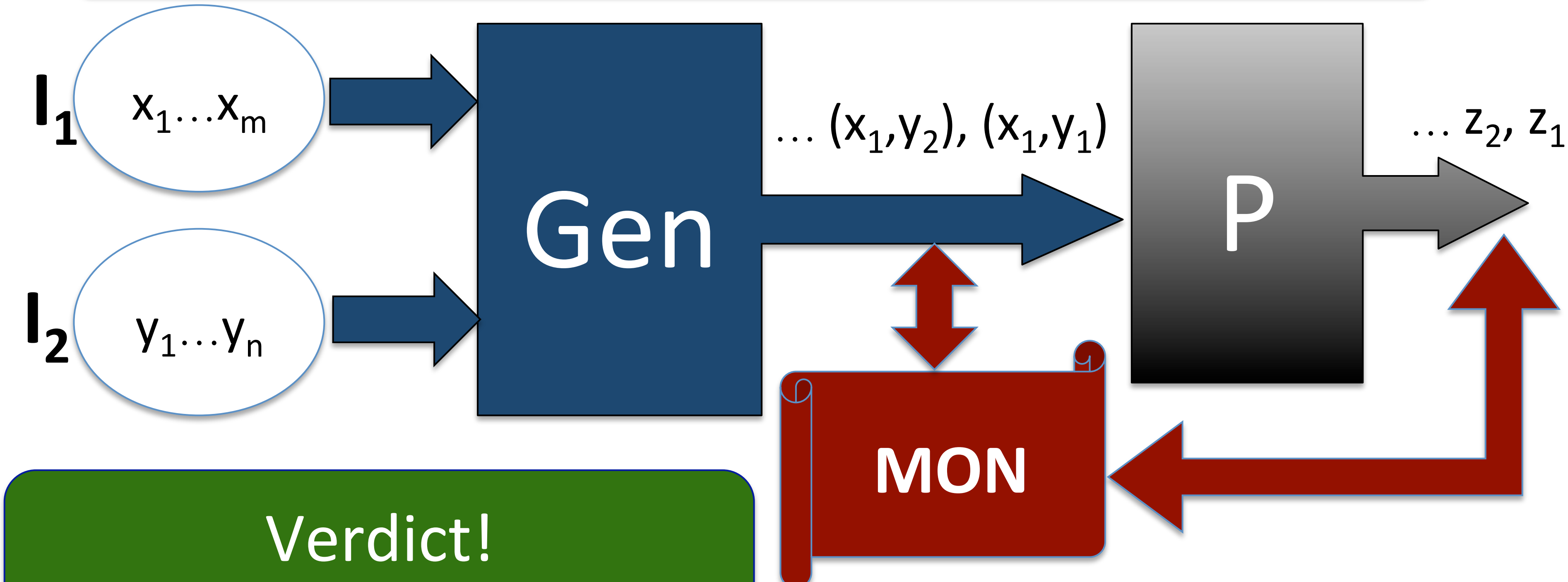
We don't extract the monitor from the property

We start with a template monitor parameterized with the concrete property

Commercial: Do not only use (regular) LTL but also **automata**

(Controlled) Offline Monitoring
Data Minimization

Assumption: Finite input domain



Verdict!
(after exhaustive generation)

Generate the minimizer!
(Black box)

Summary

- Parameterized monitor for **HyperLTL$_{2S}$**
- Online monitorability for violations of property
  - Generalizes to traces of fixed length (not only 1)
  - Order of the traces not important (they may be reordered)
- Complexity: quadratic in the length of the observed trace
- Offline monitoring under assumption of finite input domains
  - Decidable (trivial!)
  - For data minimization: extraction of a minimizer
  - Optimizations are possible (taking into account size of output domain, etc)

Acknowledgement and References

- Joint work with **Thibaud Antignac**, **Srinivas Pinisetty** and **Dave Sands**

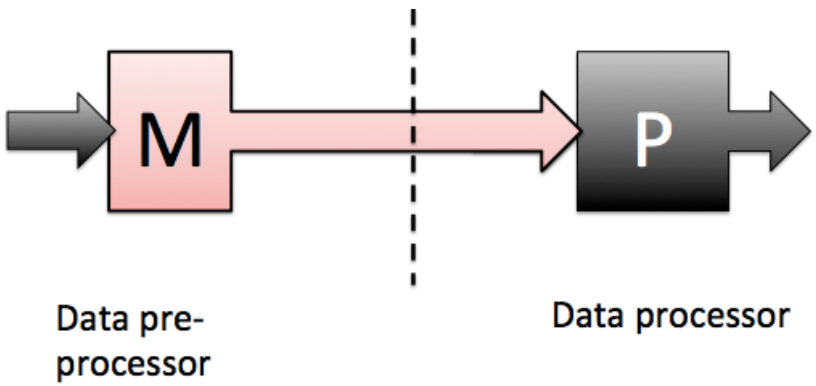
- Thibaud Antignac, David Sands, and Gerardo Schneider. *Data Minimisation: A Language-Based Approach.* In IFIP SEC'17, vol 502 of IFIP AICT, pp. 442-456, 2017
- Srinivas Pinisetty, Thibaud Antignac, David Sands, Gerardo Schneider: *Monitoring Data minimisation.* In Tech. Rep. arXiv:1801.02484v1 (Jan 2018)
- Srinivas Pinisetty, David Sands, Gerardo Schneider: *Runtime Verification of Hyperproperties for Deterministic Programs.* In FormaliSE’18 (Jun 2018)

Questions?

M is a *monolithic pre-processor* for P iff

$$P \circ M = P$$

$$M \circ M = M$$



Distributed Minimiser