Monitoring Data Minimization
And Other Similar Hyperproperties

Gerardo Schneider
Dept. of Computer Science and Engineering
Chalmers | University of Gothenburg
Sweden

ARVI Winter School
France
19-21 March 2018
“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?
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 
\textit{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...

1. `input(salary);`
2. `benefits := (salary < 10000);`
3. `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

12000
5000
90000
55000
...

False
True
False
False
...

P
Minimization

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

The minimizer M:

\[
\text{salary} \rightarrow \begin{cases} 
1000 & \text{if salary} < 10000 \\
100000 & \text{otherwise}
\end{cases}
\]
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 \( P \big|_{\text{range}(M)} \) is minimal.
Key Properties of Minimizers

Kernel of a function:

\[ \text{ker}(F) = \{ (x,y) \mid F(x) = F(y) \} \]

(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 \( \text{ker}(M) = \text{ker}(P) \)

**Theorem:**
For every \( P \) there exists a monolithic minimizer
Distributed Minimizer

But what’s the definition when $P$ receives inputs from different sources?
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.
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

\[
\begin{align*}
\text{XOR}(F,T) &= T \\
\text{XOR}(T,F) &= T \\
\text{XOR}(F,F) &= F \\
\text{XOR}(T,T) &= F \\
\end{align*}
\]

Necessarily \textit{weaker} than monolithic minimality

\[
\begin{align*}
\text{Distributed minimal} \\
\text{XOR}(F,F) \neq \text{XOR}(T,F), \\
\text{XOR}(T,T) \neq \text{XOR}(F,T), \\
\text{XOR}(F,F) \neq \text{XOR}(F,T), \\
\text{XOR}(T,T) \neq \text{XOR}(T,F) \\
\end{align*}
\]

\[
\begin{align*}
\text{But not monolithic minimal} \\
\text{XOR}(F,F) &= \text{XOR}(T,T) \\
\text{but} \\
(F,F) &\neq (T,T)
\end{align*}
\]
Distributed Minimality

Too strong a notion!

Not distributed minimal

\[ \text{OR}(F,T) = \text{OR}(T,T) \]

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 \( P(u[i \mapsto x]) \neq P(u[i \mapsto y]) \)
Distributed Minimality

\[
\begin{align*}
\text{OR } (F,F) &= F \\
\text{OR } (F,T) &= T \\
\text{OR } (T,F) &= T \\
\text{OR } (T,T) &= T
\end{align*}
\]

**Distributed minimal**

For \((F,x)\) and \((T,x)\) we can always find a value for \(x\) s.t.
\[\text{OR}(F,x) \neq \text{OR}(T,x)\] (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 $\Rightarrow$ Strong distributed $\Rightarrow$ Distributed
The Security Perspective

Attacker model: Hidden purpose ("secondary use")

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

\textit{data minimization}

and

other similar \textit{hyperproperties}
Why Monitoring?

- Program (Java source)
- Symbolic executor (KeY Project)
- Specification (JML annotations)
- Preconditions
  - States
  - Path conditions
- DataMin
  - Symbolic characterisation $\Gamma$
  - Equivalence classes $k$
  - Representatives $r$
- Sectioning
- Minimiser generation
- Solver (Z3)
- Diagnosis
- Semi-decision procedure
- Needs the source code

Cannot be applied if you want to check a third-party...
Can we do it?


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

Universal quantifiers: usually **not** monitorable

Existential quantifiers: monitorable
What Does it Mean 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...

- **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)

(90000, F) (5000, T) (1200, F)
Monitor for Data Minimization

The "intelligence" is in the OK? predicate

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i_1$</td>
<td>$o_1$</td>
</tr>
<tr>
<td>$i_2$</td>
<td>$o_2$</td>
</tr>
<tr>
<td>$i_3$</td>
<td>$o_3$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Is there a prefix with the same output and different input?
Monitor for *Monolithic Data Minimization*

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>F</td>
</tr>
<tr>
<td>5000</td>
<td>T</td>
</tr>
<tr>
<td>90000</td>
<td>T</td>
</tr>
</tbody>
</table>

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 \textit{secret} (high) and output \textit{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*ₚᵢ(ᵢ) = *P*ₚ’ᵢ(ᵢ)

\[
\forall \pi, \forall \pi' : ((\pi_{Param} \in P\text{Intrs}) \land (\pi'_{Param} \in P\text{Intrs}) \land (\pi_I = \pi'_I)) \implies (\pi_O = \pi'_O)
\]
Other properties similar to Data Minimization

<table>
<thead>
<tr>
<th>Property</th>
<th>Property expressed in $\text{Hyper}_{2S}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data minimisation (Monolithic minimality)</td>
<td>$\forall \pi, \forall \pi' : \pi_I \neq \pi'_I \Rightarrow \pi_O \neq \pi'_O$.</td>
</tr>
<tr>
<td>Non-Interference</td>
<td>$\forall \pi, \forall \pi' : (\pi_{I,L} = \pi'<em>{I,L}) \Rightarrow (\pi</em>{O,L} = \pi'_{O,L})$</td>
</tr>
<tr>
<td>Integrity</td>
<td>$\forall \pi, \forall \pi' : (\pi_{I,H} = \pi'<em>{I,H}) \Rightarrow (\pi</em>{O,H} = \pi'_{O,H})$</td>
</tr>
</tbody>
</table>
| Software doping (doping free program) | $\forall \pi, \forall \pi' :$
  \begin{align*}
  &((\pi_{\text{Param}} \in \text{PINtrs}) \land (\pi'_{\text{Param}} \in \text{PINtrs}) \land (\pi_I = \pi'_I)) \\
  \Rightarrow & \ (\pi_O = \pi'_O)
  \end{align*}$ |
| Strong distributed minimality    | $\forall \pi, \forall \pi'$,
  let $\pi_I = (i_1, \cdots, i_n), \pi'_I = (i'_1, \cdots, i'_n)$.
  
  $(\exists x \in [1, n] : i_x \neq i'_x) \land$
  
  $\forall y \in [1, n] : y \neq x \Rightarrow i_y = i'_y) \Rightarrow \pi_O \neq \pi'O$. |
Monitor for other Properties in HyperLTL$_{2S}$

Read Input $i$  
Obs. Output $o$  
Record $(i,o)$

Read Input $i$  
Obs. Output $o$  
Record $(i,o)$

Not OK?  
×

OK?  

The "intelligence" is in the OK? predicate

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i_1$</td>
<td>$o_1$</td>
</tr>
<tr>
<td>$i_2$</td>
<td>$o_2$</td>
</tr>
<tr>
<td>$i_3$</td>
<td>$o_3$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Parameterized Monitor for Properties in HyperLTL$_{2S}$

<table>
<thead>
<tr>
<th>Property</th>
<th>Property expressed in Hyper$_{2S}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data minimisation (Monolithic minimality)</td>
<td>$\forall \pi, \forall \pi' : \pi_1 \neq \pi'_1 \implies \pi_O \neq \pi'_O$.</td>
</tr>
<tr>
<td>Non-Interference</td>
<td>$\forall \pi, \forall \pi' : (\pi_{I,L} = \pi'<em>{I,L}) \implies (\pi</em>{O,L} = \pi'_{O,L})$.</td>
</tr>
<tr>
<td>Integrity</td>
<td>$\forall \pi, \forall \pi' : (\pi_{I,H} = \pi'<em>{I,H}) \implies (\pi</em>{O,H} = \pi'_{O,H})$.</td>
</tr>
<tr>
<td>Software doping (doping free program)</td>
<td>$\forall \pi, \forall \pi' : ((\pi_{Param} \in PInsts) \land (\pi'_{Param} \in PInsts) \land (\pi_1 = \pi'_1)) \implies (\pi_O = \pi'_O)$.</td>
</tr>
</tbody>
</table>
| Strong distributed minimality   | $\forall \pi, \forall \pi'$, let $\pi_1 = (i_1, \ldots, i_n), \pi' = (i'_1, \ldots, i'_n)$.

$(\exists x \in [1, n] : i_x \neq i'_x) \lor (\forall y \in [1, n] : i_y = i'_y) \implies \pi_o \neq \pi'_o$. |

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)

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 $\text{HyperLTL}_{2S}$

• 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, 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

\[ P \circ M = P \]

\[ M \circ M = M \]