Runtime Verification of Hyperproperties for Deterministic Programs

Srinivas Pinisetty, Dave Sands, Gerardo Schneider
Dept. of Computer Science and Engineering
Chalmers | University of Gothenburg
Sweden

FormaliSE@ICSE
Gothenburg, Sweden
2 June 2018
“Personal data must be: adequate, relevant, and limited to the minimum necessary in relation to the purposes for which they are processed…”

Simplifying: “You should not collect more data than what is strictly required for the intended computation”

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

* EU – 2016/679: Entered into application 25 May 2018
What is the connection with the title of our paper?

• The above “principle” is called Data Minimization

• Data minimization is a representative of an interesting class of properties

• Hyperproperties are properties defined over sets of sets of traces
  • ”Normal” properties are defined over sets of traces
Collection vs Usage: We focus on the first (restricted)

- **Previous work**: Definitions + what can be done statically

- **Goal of this work**: Monitoring data minimization and other similar hyperproperties
A Simple Program...

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

Is the information about the salary really needed?
Data Minimization (Minimality)

Definition

P is data minimal if its output is totally dependent on its inputs: any variation of input $x$ causes variation in output $y$

Two variants:

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

Monolithic case: minimality is just injectivity
Data Minimization (Minimality)

Definition

P is data minimal if its output is totally dependent on its inputs: any variation of input x causes variation in output y.

P (the “benefits” program) is not (monolithical) minimal.
Why Monitoring?

Statically detecting and ensuring (monolithic and distributed) data minimality is not easy*

Can we do it?


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...

- **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 a *trace property*

Not very important (algorithm is simpler)
Monitor for Data Minimization

The "intelligence" is in the OK? predicate

OK? (Monolithic Data Minimization)

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

<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>
Monitor for **Monolithic Data Minimization**

- Read Input \( i \)
- Obs. Output \( o \)
- Record \((i, o)\)

OK?  
Not OK? \( ✗ \)

<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 *Strong 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 (hyper)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 $P_{p}(i) = P_{p'}(i)$
Other (hyper)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>$\forall \pi, \forall \pi' : \pi_I \neq \pi'_I \implies \pi_O \neq \pi'_O.$</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>
</tbody>
</table>
| Software doping (doping free program)        | $\forall \pi, \forall \pi' :$  
  $((\pi_{\text{Param}} \in \text{PIntrs}) \land (\pi'_{\text{Param}} \in \text{PIntrs}) \land (\pi_I = \pi'_I))$  
  $\implies (\pi_O = \pi'_O)$ |
| 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 Hyperproperties in HyperLTL$_{2S}$

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

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

Not 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 Hyperproperties in HyperLTL$_{2S}$

<table>
<thead>
<tr>
<th>Property</th>
<th>Property expressed in HyperLTL$_{2S}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data minimisation (Monolithic minimality)</td>
<td>$\forall \pi, \forall \pi': \pi \neq \pi' \Rightarrow \pi_O \neq \pi'_O$.</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>
<tr>
<td>Software doping (doping free program)</td>
<td>$\forall \pi, \forall \pi': ((\pi_{\text{Param}} \in \text{PlIntrs}) \land (\pi'_{\text{Param}} \in \text{PlIntrs}) \land (\pi = \pi')) \Rightarrow (\pi_O = \pi'_O)$.</td>
</tr>
<tr>
<td>Strong distributed minimality</td>
<td>$\forall \pi, \forall i, i' \in [1, n] \forall t \in [1, n]$ $i_x \neq i'<em>x$,$\pi</em>{t,i} = \pi'_{t,i}$. $\Rightarrow \pi_0 \neq \pi'_0$.</td>
</tr>
</tbody>
</table>
Runtime Verification (Monitoring) but...

(an unimportant clarifying note)

- The RV technique we are using here doesn’t follow the “standard” way of getting the monitor

We don’t extract the monitor *from* the property

We start with a template monitor parameterized with the concrete property
(Controlled) Offline Monitoring
Data Minimization

Assumption: Finite input domain

I_1 \rightarrow x_1 \ldots x_m \rightarrow \text{Gen} \rightarrow \ldots (x_1, y_2), (x_1, y_1) \rightarrow P \rightarrow \ldots z_2, z_1

I_2 \rightarrow y_1 \ldots y_n \rightarrow \text{MON} \rightarrow \text{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)
Questions?

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

\[ P \circ M = P \]

\[ M \circ M = M \]