Epistemic Verification of Information-Flow Properties in Programs

Ioana Boureanu
Director of Surrey Centre for Cyber Security, UK

joint work @ IJCAI 2017, AAAI 2023, FM 2023, ….

with N. Gorogiannis (Facebook)
F. Raimondi (Gran Sasso Science Institute)
F. Belardinelli (Imperial College London)
V. Malvone (Télécom Paris)
F. Rajaona (Univ. of Surrey)
About me

➢ PhD in non-classical logics for (security) verification

➢ Post-doc in security and cryptography

➢ …

➢ Post-doc in security verification & provable security

➢ …

➢ Professor in secure systems

My work:

➢ Formal methods
➢ Provable Security / Formal Verification
➢ Applied Cryptography

Today: FM for non-cryptographic “privacy”
Motivation & Aim

Program-Epistemic Logics

Verification Methods of These Logics

Practical Experimentations

Conclusions
Aim

- be able to verify information-flow or privacy-like properties of concurrent programs or threads

- threads can OBSERVE certain program variables and not necessarily the same
  - Thread1 observes variable x; Thread2 observes variable y
  - But the programme does $x := y + 5$ … somewhere
  - Thread1 and Thread2 often may know the full program, or at least their program
  - So, what does Thread1 know/learn about variable y?
    - What does Thread1 know/learn about Thread2 knowing or doing something on variable y?

- This is fine… seems well-known …akin to .. non-interference, information-flow..
Aim

- Thread1 observes variable x; Thread2 observes variable y
- So, what does Thread1 know/learn about y? ...
- This is fine..., well-known even, non-interference, information-flow..
- But, ..
- NOT for “high-level” programs OR
- NOT expressive in the sense meant where… “what does Thread1 learn … about Thread2 doing/knowing…?”
- Logic formulae expressing properties about program states: e.g.,

  “Thread1 knows that variable x is equal to y + 5”
  “Thread2 does not know that variable x is equal to y + 5”
What expressivity we mean?

- **epistemic logics**, i.e., logics of knowledge – “knowing logical facts” → expressions of rich properties (e.g., information flow, non-interference)

- well-used in verification of general-purpose concurrent & distributed SYSTEMS (e.g., Byzantine agreement) via epistemic model checkers such as MCMAS, Verics, MCK, etc....
epistemic logics well-used in systems’ model checkers
systems BUT...

► these are NOT epistemic specifications on programs (like we mean here)

► it is hard to capture rich (e.g., first-order) state specifications,
  since the base logic of most epistemic verifiers is *propositional*
... meanwhile, base logics of programs are *VERY expressive*

► predicate transformers (e.g., weakest precondition) are used to reduce
  verification to FO queries to SMT solvers ...i.e., away from model-checking
Back to our aim

- be able to verify **information-flow** or **non-interference** properties of concurrent programs or threads, under **their partial observability**

- Focus on rich epistemic properties over program states: e.g.,

  “Thread1 knows that when program C will execute Thread2 knows variable x is equal to y + 5”

- Q: Can we harness SMT solving or shall we rely on epistemic model checking?
Motivation & Aim

Program-Epistemic Logics

Verification Methods of These Logics

Practical Experimentations

Conclusions
Syntax

- $A$  
  a finite set of *threads* or program-observers

- $V$  
  a countable set of variables

- $p \subseteq V$  
  a non-empty set of *program variables*

- $o_A \subseteq p$  
  the variables the thread $A \in A$ can *observe*

- $n_A = p \setminus o_A$  
  variables thread $A \in A$ *cannot observe*
Syntax

First Epistemic Language $L_K$

$[IJCAI'17]$

$\mathbf{L_{QF}}$

base language = a quantifier-free, FO language

$\mathbf{L_{FO}}$

extension of $L_{QF}$ with quantifiers

$$\phi ::= \pi | \neg \phi | \phi_1 \land \phi_2 | \phi_1 \lor \phi_2 | \phi_1 \Rightarrow \phi_2 | \forall x. \phi | \exists x. \phi$$

$\mathbf{L_K}$

extension of $L_{QF}$ with epistemic modalities $K_A$

$$\alpha ::= \pi | \neg \alpha | \alpha_1 \land \alpha_2 | \alpha_1 \lor \alpha_2 | \alpha_1 \Rightarrow \alpha_2 | K_A \alpha$$
First Program-Epistemic Specifications $L □ K$

- $C$ a (possibly infinite) set of *commands*
- $L □ K$ extends $L_K$ with every formula $\beta = □_C \alpha$, meaning “at all final states of $C$, $\alpha$ holds”

**Example**

“at the end of the vote-counting, a partial-observing thread $thread1$ (who can see certain aspects of the program) does not know that voter 1 vote for candidate 1”:

$$\Box_{E_{VotingProgram}} \neg K_{thread1} V_{1,1}$$

where $V_{1,1}$ is a formula in $L_{QF}$ which here is linear integer arithmetic.
First-order Semantics

- state $s : \mathcal{V} \to \mathcal{D}$.
- set of all states $\mathcal{U}$

\[
\begin{align*}
s \models \pi & \quad \iff \quad \text{in accordance to interpretation } I \\
 s \models \phi_1 \circ \phi_2 & \quad \iff \quad (s \models \phi_1) \circ (s \models \phi_2) \\
s \models \neg \phi & \quad \iff \quad s \not\models \phi \\
s \models \exists x. \phi & \quad \iff \quad \exists c \in \mathcal{D}. \, s[x \mapsto c] \models \phi \\
s \models \forall x. \phi & \quad \iff \quad \forall c \in \mathcal{D}. \, s[x \mapsto c] \models \phi.
\end{align*}
\]

where $\circ$ is $\land$, $\lor$ or $\Rightarrow$, and $I$ is an interpretation of constants, functions and predicates in $\mathcal{L}_{\text{QF}}$ over the domain $\mathcal{D}$.

The **interpretation** $[\phi]$ of a first-order formula $\phi$ is the set of states satisfying it, i.e., $[\phi] = \{ s \in \mathcal{U} \mid s \models \phi \}$.
Towards a Program-Epistemic Semantics

- **Indistinguishability relation** $\sim_X$ over states

  $s \sim_X s' \iff \forall x \in X. (s(x) = s'(x))$

  where $X \subseteq \mathcal{V}$

- **Transition relation (over states)** of any command $C$

  $$R_C(s) = \{ s' \mid (s, s') \in R_C \} \quad R_C(W) = \bigcup_{s \in W} R_C(s)$$

- **Strongest postcondition operator** is a partial function

  $SP(\_ , -) : \mathcal{L}_{FO} \times \mathcal{C} \rightarrow \mathcal{L}_{FO}$

  $SP(\phi, C) = \psi \iff \llbracket \psi \rrbracket = R_C(\llbracket \phi \rrbracket)$
Interpretation of a program specification $\beta$

The satisfaction relation $W, s \models \beta$

$$W, s \models \pi \iff s \models \pi$$

$$W, s \models \neg \alpha \iff W, s \not\models \alpha$$

$$W, s \models \alpha_1 \circ \alpha_2 \iff (W, s \models \alpha_1) \circ (W, s \models \alpha_2)$$

$$W, s \models K_A \alpha \iff \forall s' \in W. (s \sim_o A s' \implies W, s' \models \alpha)$$

$$W, s \models \Box_C \alpha \iff \forall s' \in R_C(s). (R_C(W), s' \models \alpha)$$

where $\circ$ is $\land$, $\lor$, or $\implies$, and $C \in C$ is a command.

- **Validity of program specifications $\phi \models \beta$**
  for all $s \in \llbracket \phi \rrbracket$, we have that $\llbracket \phi \rrbracket, s \models \beta$.

  $\phi \models K_A \pi$ means that in all states satisfying $\phi$, thread A knows $\pi$

  $\phi \models \Box_C \neg K_A \pi$ means that if command C starts at a state satisfying $\phi$, then in all states where the execution finishes, thread A does not know $\pi$
Motivation & Aim

Program-Epistemic Logics

Verification Methods of These Logics

Practical Experimentations

Conclusions
First Reduction to First-Order Validity

- Validity of program specifications $\phi \models \beta$
  for all $s \in [\phi]$, we have that $[\phi], s \models \beta$.

- Recall: strongest postcondition operator is a partial function $SP(\cdot, \cdot) : \mathcal{L}_{FO} \times \mathcal{C} \rightarrow \mathcal{L}_{FO}$

  $SP(\phi, C) = \psi \iff [\psi] = R_C([\phi])$

If the strongest postcondition operator is computable for the chosen base logic/programming language, then validity of program-epistemic specifications reduces to validity in first-order fragments (such as QBF and Presburger arithmetic).

The translation $\tau : \mathcal{L}_K \rightarrow \mathcal{L}_{FO}$ of epistemic formulas into the first-order language

$$
\begin{align*}
\tau(\phi, \pi) &= \pi \\
\tau(\phi, \neg \alpha) &= \neg \tau(\phi, \alpha) \\
\tau(\phi, \alpha_1 \circ \alpha_2) &= \tau(\phi, \alpha_1) \circ \tau(\phi, \alpha_2) \\
\tau(\phi, K_A \alpha) &= \forall n_A . (\phi \Rightarrow \tau(\phi, \alpha))
\end{align*}
$$

Recall my question re model checking vs SMT solving?
Motivation & Aim

Program-Epistemic Logics

Verification Methods of These Logics

Practical Experimentations

Conclusions
Loop-Free Example Programming Language

<table>
<thead>
<tr>
<th>Command $C$</th>
<th>$SP(\phi, C)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x := *$</td>
<td>$\exists y. \phi[y/x]$</td>
</tr>
<tr>
<td>$x := e$</td>
<td>$\exists y. (x = e[y/x] \land \phi[y/x])$</td>
</tr>
<tr>
<td>$\text{if}(\pi) C_1 \text{else} C_2$</td>
<td>$SP(\pi \land \phi, C_1) \lor SP(\neg \pi \land \phi, C_2)$</td>
</tr>
<tr>
<td>$C_1; C_2$</td>
<td>$SP(SP(\phi, C_1), C_2)$</td>
</tr>
</tbody>
</table>

where $x$ is a program variable and $y$ is a fresh logical variable.

- $SP(-, -)$ may only introduce existential quantifiers.
- If $x \notin FV(\phi)$, then $SP(\phi, x := e) = (\phi \land x = e)$. That is, if $x$ is unrestricted, no quantifiers are introduced.
- For a fixed $C$, the size of $SP(\phi, C)$ is polynomial in $\|\phi\|$.

- Enough to express .. somewhat… simple communication protocols, anonymity-driven systems, knowledge proofs…
Three Ballot Voting

- for a candidate, exactly two atomic ballots.
- against a candidate, exactly one atomic ballot.

Here:
- Vote privacy
- No active attacker
Three Ballot Specifications

- $m > 2$ candidates
  $n > 2$ voters
- $\mathbf{L_{QF}}$ linear integer arithmetic
- Threads $A = \{1, \ldots, n; P\}$: voters + $P$ is a ‘public observer’/ general program
- Program variables
  \[ p = \bigcup_{j=1}^{m} \{c_j\} \cup \bigcup_{i=1}^{n} \bigcup_{j=1}^{m} \bigcup_{k=1}^{3} \{b_{ijk}\} \]
  \[ o_i = \bigcup_{j=1}^{m} \{c_j\} \cup \bigcup_{j=1}^{m} \bigcup_{k=1}^{3} \{b_{ijk}\} \]
  \[ o_P = \bigcup_{j=1}^{m} \{c_j\} \]
  \[ n_i = p \setminus o_i \]
- Non-observable variables
- Vote Counting (the number of ticks voter $i$ has entered for candidate $j$)
  \[ S_{i,j} = \sum_{k=1}^{3} b_{ijk} \]
- $\mathbf{L_{QF}}$ Presburger arithmetic
- $c_j$ total number of atomic-ballot ticks for candidate $j$
- $b_{ijk}$ if voter $i$ ticked next to candidate $j$ on the $k$-th atomic ballot

\[ c_1 := \sum_{i=1}^{n} S_{i,1} \; ; \; \ldots \; ; \; c_m := \sum_{i=1}^{n} S_{i,m} \]
Three Ballot Specifications (cont’d)

- Macros to model the protocol

\[ S_{i,j} \equiv \sum_{k=1}^{3} b_{ijk} \]
\[ B \equiv \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{m} \bigwedge_{k=1}^{3} (b_{ijk} = 0 \lor b_{ijk} = 1) \]
\[ V_{i,j} \equiv (S_{i,j} = 2) \]
\[ \bar{V}_{i,j} \equiv (S_{i,j} = 1) \]
\[ CV_{i}^{\geq 0} \equiv \bigvee_{j=1}^{m} V_{i,j} \]
\[ CV_{i}^{\leq 1} \equiv \bigwedge_{j=1}^{m} \left( V_{i,j} \Rightarrow \bigwedge_{j' = 1, j' \neq j}^{m} \bar{V}_{i,j'} \right) \]
\[ CV \equiv \bigwedge_{i=1}^{n} (CV_{i}^{\geq 0} \land CV_{i}^{\leq 1}) \]
\[ NU \equiv \bigwedge_{j=1}^{m} \bigvee_{i=1}^{n} V_{i,j} \]
\[ NU_{mod} i \equiv \bigwedge_{j=1}^{m} \bigvee_{i'=1, i' \neq i}^{n} V_{i',j} \]
\[ I \equiv B \land CV \land NU \]
\[ I_{mod} i \equiv B \land CV \land NU_{mod} i \]

\[ SP(I, C) = I \land (c = (\sum_{i=1}^{n} S_{i,1}, \ldots, \sum_{i=1}^{n} S_{i,m})) \]

**c** is the tuple \((c_1, \ldots, c_m)\)
Three Ballot Specifications (cont’d)

\[ \alpha_1 = \neg K_P V_{1,1} \]
the observer P does not know that voter 1 voted for candidate 1

\[ \alpha_2 = \neg K_1 V_{2,1} \]
voter 1 does not know that voter 2 voted for candidate 1

Vote Privacy Verification

\[ I \models \Box_C \alpha_1 \]

\[ I_{\text{mod} 1} \models \Box_C \alpha_2 \]

\[ I \not\models \Box_C \alpha_2. \]

\[ SP(I, C) = I \land (c = (\sum_{i=1}^{n} S_{i,1}, \ldots, \sum_{i=1}^{n} S_{i,m})) \]

\[ \text{translation of K formulae} \]

\[ \Rightarrow \text{Presburger formulae} + \]
Experimental Results (on a simple laptop)

The figure shows a graph plotting time (in seconds) against the number of voters. The graph has a logarithmic scale on the y-axis and a linear scale on the x-axis. Different lines represent different scenarios or settings, indicated by symbol markers. The x-axis is labeled "Number of voters" and the y-axis is labeled "Time (sec)." The graph illustrates how the time taken increases with the number of voters.

For example, the legend includes symbols for $m = 2$, $m = 3$, and $m = 5$, with corresponding curves indicating how the time varies with different numbers of voters. The graph suggests that the time required for processing increases significantly as the number of voters increases, especially when compared to a simple laptop environment.
(i) MCMAS is faster, or equally fast, for $n \leq 7$, but slower for all $n > 7$;
(ii) we can be faster than MCMAS by a factor of $> 100$ (e.g., when $n = 32$) when checking $\alpha_1$, whilst when verifying $\alpha_3$ our speed-up is of several orders of magnitudes.

Recall my question re model checking vs SMT solving?
So, where are we?

- 😊 we “played” with some logics, .. We gave program-epistemic specifications, expressing requirements that given epistemic properties hold on all final states of the program

- 😞 we have an efficient method of reducing the validity of program-epistemic specifications to appropriate queries to SMT solvers

- 😞 space for improvements...

... epistemic $K_A$ operator can appear only after program $\square_C$ operator..., we cannot have $K_A K_B \phi$ .. , meaning we cannot have more than one agent “knowing”
Motivation & Aim

Program-Epistemic Logics

Verification Methods of These Logics

Practical Experimentations

Conclusions
If we want the program operator and the epistemic operator to commute, perhaps *link the program language and the logic more?*

Programs, e.g., assignments, leak information; perhaps, we can model this program “leak” via logics: *announcement logics* [Plaza’89]

**Ali-Baba’s Cave Zero Knowledge**

- Peggy randomly takes either path A or B, while Victor waits outside.
- Victor chooses an exit path.
- Peggy reliably appears at the exit Victor names.

Peggy
- announces “success on path $x_1$”
- announces “success on path $x_2$”
- announces “success on path $x_3$”
Second Program-Epistemic Language

perhaps link the program language and the logic more?

Announcement logics [Plaza '89] ...

Program Syntax

\[
P ::= \alpha? \quad \text{(test/announcement)}
\]

\[
x_G := e \quad \text{(assignment)}
\]

\[
\text{new}k_G \cdot P \quad \text{(declare } k \text{ visible to } G)
\]

\[
P; Q \quad \text{(sequential composition)}
\]

\[
P \sqcap Q \quad \text{(nondeterministic choice)}
\]

Second Epistemic Logic Syntax \( \mathcal{L}_P \)

\[
\alpha ::= \pi \quad \text{(atomic predicate)}
\]

\[
\alpha \land \alpha' \quad \text{(conjunction)}
\]

\[
\neg \alpha \quad \text{(negation)}
\]

\[
K_A \alpha \quad \text{(knowledge modality)}
\]

\[
[\alpha']^G \alpha \quad \text{(public announcement formula)}
\]

\[
\forall x_G \cdot \alpha \quad \text{(universal quantification)}
\]
Let’s re-think relational semantics (for the new $L_P$...) 

- $R(v := x; v := 0, \omega) = R(v := 0, \omega)$
  (wrong if the thread knows the program)

- $wp(v := x, \alpha) = \alpha[v \setminus x]$

**Example**

$x \in \{0, 1\}$, $v$ is visible, and $x$ a secret

Does the program $P = v := x$ leaks the secret $x$?

$$wp(v := x, K(x = 0) \lor K(x = 1)) = K(x = 0) \lor K(x = 1)[v \setminus x]$$

True.

What if the program $P = (v := x \land v := \neg x)$?

depends on the thread’s observability of program execution
Relational Semantics for $\mathcal{L}_P$.

So, it depends on a few things and it is not obvious

For public programs, …

$$R_W(P \sqcap Q, s) = \{s'[c_{Ag} \mapsto l] \mid s' \in R_W(P, s)\}$$
$$\cup \{s'[c_{Ag} \mapsto r] \mid s' \in R_W(Q, s)\}$$

$$R_W(P; Q, s) = \bigcup_{s' \in R_W(P, s)} \{R_{R_W}^*(P, W)(Q, s')\}$$

$$R_W(x_G := e, s) = \{s[k_G \mapsto s(x_G), x_G \mapsto s(e)]\}$$

$$R_W(\text{new}k_G \cdot P, s) = R_W^*(P, \{s[k_G \mapsto d] \mid d \in D\})$$

$$R_W(\beta?, s) = \text{if } (W, s) \models \beta \text{ then } \{s\} \text{ else } \emptyset$$
Second, More Expressive Program-Epistemic Language

Program-Epistemic Logic $\mathcal{L}_{PK}$

$$\alpha ::= \pi \mid \alpha \land \alpha' \mid \neg \alpha \mid K_a\alpha \mid [\alpha']\alpha \mid \forall x_G \cdot \alpha \mid \Box_P \alpha$$

- $\Box_P(K_v(secret \mod 2))$
- $K(\Box_Psecret \mod 2 = 0)$
- $\Box_{DC}(K_0(x \Leftrightarrow \bigvee_{i=0}^{n-1} p_i))$

$(W, s) \models [\beta]\alpha$ iff $(W, s) \models \beta$ implies $(W|_\beta, s) \models \alpha$

$(W, s) \models \Box_P\alpha$ iff for all $s' \in R_W(P, s)$, $(R^*_W(P, W), s') \models \alpha$

$(W, s) \models \forall x_G \cdot \alpha$ iff for all $c \in D$, $(\bigcup_{d \in D}{s'[x_G \mapsto d] \mid s' \in W}, s[x_G \mapsto c]) \models \alpha$
Program-based Semantics for $\mathcal{L}_K$...

Linking programs and formula “tighter” than in the first attempt

$\text{wp} : \mathcal{L}_P \times \mathcal{L}_K \rightarrow \mathcal{L}_K$

$\text{wp}(P \sqcap Q, \alpha) = \text{wp}(P, \alpha) \lor \text{wp}(Q, \alpha)$

$\text{wp}(P ; Q, \alpha) = \text{wp}(P, \text{wp}(Q, \alpha))$

$\text{wp}(x_G := e, \alpha) = \forall k_G \cdot [k_G = e](\alpha[x_G \setminus k_G])$

$\text{wp}(\text{new} k_G \cdot P, \alpha) = \forall k_G \cdot \text{wp}(P, \alpha)$

$\text{wp}(\beta?, \alpha) = \lbrack \beta \rbrack \alpha$

Relational semantics at states and this WP-based semantics at formulae coincide
Motivation & Aim

Program-Epistemic Logics

Verification Methods of These Logics

Practical Experimentations

Conclusions
Main theorem

- $\models \phi$ a set of states satisfying FO formula $\phi$
- $\alpha \in \mathcal{L}_{PK}$

$$\models \phi \models \alpha \iff \text{FO formula } \phi \land \neg \tau(\phi, \alpha) \text{ unsatisfiable}$$

where $\tau: \mathcal{L}_{FO} \times \mathcal{L}_{PK} \to \mathcal{L}_{FO}$

$$\tau(\phi, \pi) = \pi \quad \tau(\phi, K_a \alpha) = \forall n \cdot (\phi \rightarrow \tau(\phi, \alpha))$$
$$\tau(\phi, \neg \alpha) = \neg \tau(\phi, \alpha) \quad \tau(\phi, [\beta] \alpha) = \tau(\phi, \beta) \rightarrow \tau(\phi \land \tau(\phi, \beta), \alpha)$$
$$\tau(\phi, \alpha_1 \circ \alpha_2) = \tau(\phi, \alpha_1) \circ \tau(\phi, \alpha_2) \quad \tau(\phi, \Box_P \alpha) = \tau(\phi, \wp(P, \alpha))$$
$$\tau(\phi, \forall x \cdot \alpha) = \forall x \cdot \tau(\phi, \alpha)$$

One “go” translation for the “full” logic, unlike before
Main theorem

• \[\models \phi\] a set of states satisfying FO formula \(\phi\)
• \(\alpha \in \mathcal{L}_{PK}\)

\[\models \phi\] \(\models\) \(\alpha \Leftrightarrow\) FO formula \(\phi \wedge \neg \tau(\phi, \alpha)\) unsatisfiable

Mechanised the translation in Haskell

```haskell
27 tau :: ModalFormula -> Formula a -> ModalFormula
28 tau phi (Atom p) = Atom p
29 tau phi (Neg alpha) = Neg (tau phi alpha)
30 tau phi (Conj as) = Conj [tau phi a | a <- as]
31 tau phi (Disj as) = Disj [tau phi a | a <- as]
32 tau phi (Imp alpha1 alpha2) = tau phi alpha1 = tau phi alpha2
33 tau phi (Equiv alpha1 alpha2) = (tau phi (alpha1 = alpha2)) \wedge (tau phi (alpha2 = alpha1))
34 tau phi (K ag alpha) = mkForAll (nonobs ag) (phi \rightarrow tau phi alpha)
35 tau phi (Box p alpha) = tau phi (wp alpha p)
36 tau phi (ForAllB n alpha) = ForAllB n (tau phi alpha)
38 tau phi (ExistsB n alpha) = ExistsB n (tau phi alpha)
39 tau phi (ForAllI n d alpha) = ForAllI n d (tau phi alpha)
40 tau phi (ExistsI n d alpha) = ExistsI n d (tau phi alpha)
```

\(\mathcal{L}_{PK}\) Model Checking as First-Order (Un)satisfiability
Model Checking as First-Order (Un)satisfiability

Experiments before (knowledge-based information flow in programs for voting, anonymous communication, …, ), BUT more expressive and a bit slower

<table>
<thead>
<tr>
<th>n</th>
<th>$\tau_{wp + Z3}$</th>
<th>$\tau_{SP + Z3}$</th>
<th>$\tau_{wp + CVC5}$</th>
<th>$\tau_{SP + Z3}$</th>
<th>$\tau_{wp + Z3}$</th>
<th>$\tau_{SP + Z3}$</th>
<th>$\tau_{wp + Z3}$</th>
<th>$\tau_{SP + Z3}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.05 s</td>
<td>4.86 s</td>
<td>0.01 s</td>
<td>0.01 s</td>
<td>0.01 s</td>
<td>0.01 s</td>
<td>0.01 s</td>
<td>N/A</td>
</tr>
<tr>
<td>50</td>
<td>31 s</td>
<td>t.o.</td>
<td>0.41 s</td>
<td>0.05 s</td>
<td>0.06 s</td>
<td>0.03 s</td>
<td>0.02 s</td>
<td>N/A</td>
</tr>
<tr>
<td>100</td>
<td>t.o.</td>
<td>t.o.</td>
<td>3.59 s</td>
<td>0.15 s</td>
<td>0.16 s</td>
<td>0.07 s</td>
<td>0.06 s</td>
<td>N/A</td>
</tr>
<tr>
<td>200</td>
<td>t.o.</td>
<td>t.o.</td>
<td>41.90 s</td>
<td>1.27 s</td>
<td>0.71 s</td>
<td>0.30 s</td>
<td>0.20 s</td>
<td>N/A</td>
</tr>
</tbody>
</table>

… (“SP” stands for the previous method at IJCAI17)
So, why and ...are we done?

How come we do not depreciate so much in efficiency, even *if we allow* $K_a K_b \phi$ *and operator* $K$ *even in front of operator* $\Box_c$?

- public announcement $\rightarrow$ model update/shrinking 😊

How come we can allow the program operate and the K operator to commute?

- Single assignment of variables ..!! 😞
Motivation & Aim

Program-Epistemic Logics

Verification Methods of These Logics

Practical Experimentations

Conclusions
Similar to the ones you saw (perhaps a “mix” of the two), but
no public announcements
the programs are modelled with dynamic logics [Vardi2013]
Assignments different via substitutions

Logic
\[
\alpha ::= \pi \mid \neg \alpha \mid \alpha \land \alpha \mid (K_\alpha \alpha)[\vec{x}/\vec{e}] \mid [\rho]\alpha \\
\rho ::= x := e \mid \phi?
\]

\[
(W, s) \models (K_\alpha \alpha)[\vec{x}/\vec{e}] \iff \text{for all } s' \in W, \\
s' \sim_\alpha s[\vec{x} \mapsto s(\vec{e})] \text{ implies } \\
(W, s') \models \alpha
\]

\[
(W, s) \models [\rho]\alpha \iff \text{for all } s' \in R_\rho(s), (R_\rho(W), s') \models \alpha
\]

We get derived dynamic operators ..
\[
[\rho; \rho'] \alpha ::= [\rho] [\rho'] \alpha \\
[\rho \sqcup \rho'] \alpha ::= [\rho] \alpha \lor [\rho'] \alpha
\]
Motivation & Aim

Program-Epistemic Logics

Verification Methods of These Logics

Practical Experimentations

Conclusions
Related Work

We defined a rich program-epistemic logic (mixing a dynamic and epistemic logic in a way that can be used to handle many-sided and semi-public environments) and tested it against a number of use-cases. Indeed, our translation from our epistemic-program logic to a propositional one can be reduced to SMT-solving. The value of our methodology is that, for instance, the satisfiability problem of a given epistemic-program logic formula can be reduced to SMT-solving even in front of operator $\Box_c$. Why this methodology.

Performances on Verifying the Dining-cryptographers Problem

More expressive than IJCAI 2017 --> we allow $K_aK_b\phi$ and operator $K$ even in front of operator $\Box_c$.

Still faster than model checking.
## Yet Another Program-Epistemic Logics

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>IJCAI 2017</th>
<th>AAAI 2023</th>
<th>FM 2023</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$K$ possible before $[prog]$</td>
<td>☹ no</td>
<td>☺ yes</td>
<td>☺ yes</td>
</tr>
<tr>
<td>2</td>
<td>only one agent</td>
<td>☹ yes</td>
<td>☺ no</td>
<td>☺ no</td>
</tr>
<tr>
<td>3</td>
<td>program public</td>
<td>☹ no</td>
<td>NaN</td>
<td>☺ yes</td>
</tr>
<tr>
<td>4</td>
<td>announcements</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>5</td>
<td>multiple assignments</td>
<td>☺ yes</td>
<td>☺ yes</td>
<td>☺ no</td>
</tr>
<tr>
<td>6</td>
<td>efficiency</td>
<td>x</td>
<td>☹ $2^x$</td>
<td>☺ x (due to SSA)</td>
</tr>
</tbody>
</table>
Motivation & Aim

Program-Epistemic Logics

Verification Methods of These Logics

Practical Experimentations

Conclusions
Take-Home Message

• Programming languages and logics to model threads
• with each “reasoning” on values/knowledge/facts

• Program and logic semantics that models “intelligent” threads

• Good for privacy/information-flow/rich non-interference properties

• Model checking delegated to SMT-solvers via translations to FO

• Implemented in Haskell here: [https://github.com/UoS-SCCS/program-epistemic-logic-2-smt](https://github.com/UoS-SCCS/program-epistemic-logic-2-smt)

• Applied in the papers I spoke of to 3BV, dinning cryptographers, logic puzzles;

• WIP: applied to fault tolerance protocols, an emulation of Uber booking, ZK proof (Ali-Baba), membership proofs...
Conclusions & Future Work

... 

- We played with a few program-expressing logics with privacy/observability purposes

Future Work

- Beyond public action/perfect recall: private actions and bounded recall

- Probabilistic programs, loops
Thank you

... for listening....

i.boureanu@surrey.ac.uk

*Images are copyrighted as per their source; pls. do not distribute without checking