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 → Imperial College London
- Post-doc in security and cryptography → ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE
- …
- Post-doc in security verification & provable security → Imperial College London
- …
- Professor in secure systems --> UNIVERSITY OF SURREY, Surrey Centre for Cyber Security (SCCS)

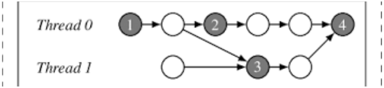
My work:

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

Today: FM for non-cryptographic "privacy"

Aim

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



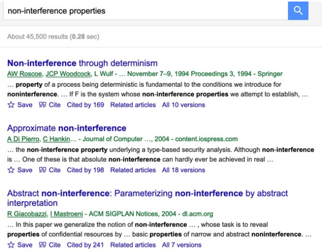
- threads can OBSERVE certain program variables and not necessarily the same
  - Thread1 observes variable x; Thread2 observes variable y
  - But the programme does x:= y+ 5 … somewhere
  - Thread1 and Thread2 *often may* know the full program, or *at least* their program
  - So, **what** does Thread1 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 …*

  aboutThread2 doing/knowing…?"

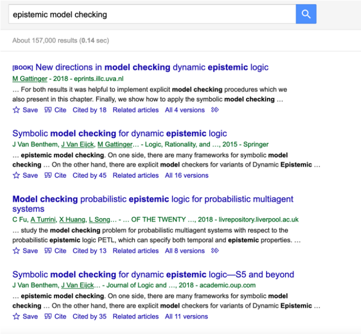
- Logic formulae expressing properties about program states: e.g.,

  "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 model checking

About 157,000 results (0.14 sec)

[BOOK] New directions in model checking dynamic epistemic logic
M Gattinger - 2018 - eprints.illc.uva.nl
… For both results it was helpful to implement explicit model checking procedures which we also present in this chapter. Finally, we show how to apply the symbolic model checking …
Save Cite Cited by 18 Related articles All 4 versions

Symbolic model checking for dynamic epistemic logic
J Van Benthem, J Van Eijck, M Gattinger… - Logic, Rationality, and …, 2015 - Springer
… epistemic model checking. On one side, there are many frameworks for symbolic model checking … On the other hand, there are explicit model checkers for variants of Dynamic Epistemic …
Save Cite Cited by 45 Related articles All 16 versions

Model checking probabilistic epistemic logic for probabilistic multiagent systems
C Fu, A Turrini, X Huang, L Song… - … OF THE TWENTY …, 2018 - livrepository.liverpool.ac.uk
… study the model checking problem for probabilistic multiagent systems with respect to the probabilistic epistemic logic PETL, which can specify both temporal and epistemic properties. …
Save Cite Cited by 13 Related articles All 8 versions

Symbolic model checking for dynamic epistemic logic—S5 and beyond
J Van Benthem, J Van Eijck… - Journal of Logic and …, 2018 - academic.oup.com
… epistemic model checking. On one side, there are many frameworks for symbolic model checking … On the other hand, there are explicit model checkers for variants of Dynamic Epistemic …
Save Cite Cited by 35 Related articles All 11 versions

Hmmm ...

- epistemic logics well-used in systems' model checkers systems BUT...

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

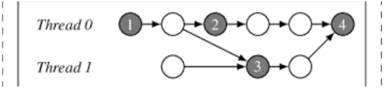
- :(it is hard to capture rich (e.g., first-order) state specifications, since the base logic of most epistemic verifiers is *propositional*

... 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

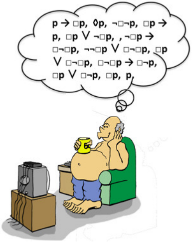
Back to our aim

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

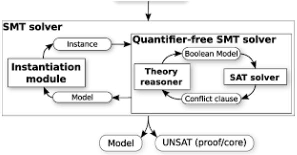


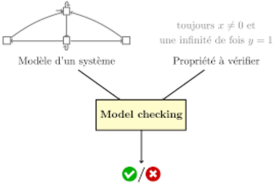
- Focus on rich epistemic properties over program states: e.g.,

"Thread1 knows that when program C will executeThread2 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

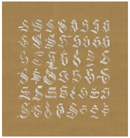
Conclusions

- $\mathcal{A}$ — a finite set of *threads* or program-observers
- $\mathcal{V}$ — a countable set of variables
- $\mathbf{p} \subseteq \mathcal{V}$ — a non-empty set of *program variables*
- $\mathbf{o}_A \subseteq \mathbf{p}$ — the variables the thread $A \in \mathcal{A}$ can *observe*
- $\mathbf{n}_A = \mathbf{p} \setminus \mathbf{o}_A$ — variables thread $A \in \mathcal{A}$ *cannot observe*

Syntax

First Epistemic Language L_K

[IJCAI'17]



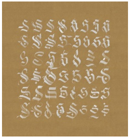
- L_{QF} — *base language* = a quantifier-free, FO language
- L_{FO} — extension of L_{QF} with quantifiers

$$\varphi ::= \pi \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \Rightarrow \varphi_2 \mid \forall x.\, \varphi \mid \exists x.\, \varphi$$

- L_K — extension of L_{QF} with epistemic modalities K_A

$$\alpha ::= \pi \mid \neg\alpha \mid \alpha_1 \wedge \alpha_2 \mid \alpha_1 \vee \alpha_2 \mid \alpha_1 \Rightarrow \alpha_2 \mid K_A\alpha$$

First Program-Epistemic Specifications $L_{\Box K}$



- C a (possibly infinite) set of *commands*
- $L_{\Box K}$ extends L_K with every formula $\beta = \Box_C \alpha$, meaning "*at <u>all</u> final states of C, α 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*":*

$$\Box_{EVotingProgram} \neg K_{thread1} V_{1,1}$$

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

First-order Semantics

Looking for

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

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

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

The *interpretation* $[\![\phi]\!]$ of a first-order formula ϕ 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

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

  - *strongest postcondition operator* is a partial function $SP(-,-) : \mathcal{L}_{\mathrm{FO}} \times \mathcal{C} \rightharpoonup \mathcal{L}_{\mathrm{FO}}$

$$SP(\phi, C) = \psi \quad \text{iff} \quad [\![\psi]\!] = R_C([\![\phi]\!])$$

Interpretation of a program specification β

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

$$
\begin{array}{lcl}
W, s \Vdash \pi & \iff & s \models \pi \\
W, s \Vdash \neg\alpha & \iff & W, s \nVdash \alpha \\
W, s \Vdash \alpha_1 \circ \alpha_2 & \iff & (W, s \Vdash \alpha_1) \circ (W, s \Vdash \alpha_2) \\
W, s \Vdash \mathrm{K}_A\alpha & \iff & \forall s' \in W.\, (s \sim_{\mathbf{o}_A} s' \implies W, s' \Vdash \alpha) \\
W, s \Vdash \Box_C\alpha & \iff & \forall s' \in R_C(s).\, (R_C(W), s' \Vdash \alpha)
\end{array}
$$

where $\circ$ is $\wedge$, $\vee$, or $\Rightarrow$, and $C \in \mathcal{C}$ is a command.

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

$\phi \Vdash \mathrm{K}_A\pi$ means that in all states satisfying ϕ, thread A knows π

$\phi \Vdash \Box_C\neg\mathrm{K}_A\pi$ means that if command C starts at a state satisfying ϕ, then in all states where the execution finishes, thread A does not know π

First Reduction to First-Order Validity

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

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

$$SP(\phi, C) = \psi \quad \text{iff} \quad [\![\psi]\!] = R_C([\![\phi]\!])$$

Recall my question re model checking vs SMT solving?

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).

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

$$\tau(\phi, \pi) = \pi \qquad \tau(\phi, \alpha_1 \circ \alpha_2) = \tau(\phi, \alpha_1) \circ \tau(\phi, \alpha_2)$$
$$\tau(\phi, \neg\alpha) = \neg\tau(\phi, \alpha) \qquad \tau(\phi, \mathsf{K}_A \alpha) = \forall \mathbf{n}_A. (\phi \Rightarrow \tau(\phi, \alpha))$$

Motivation & Aim

Program-Epistemic Logics

Verification Methods of These Logics

Practical Experimentations

Conclusions

Loop-Free Example Programming Language

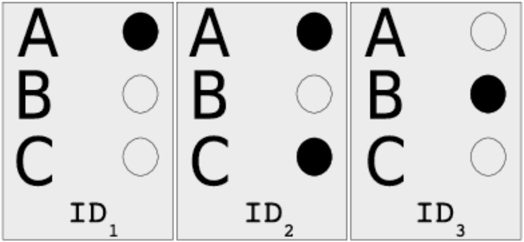
Command C	$SP(\phi, C)$
$x := *$	$\exists y.\, \phi[y/x]$
$x := e$	$\exists y.\, (x = e[y/x] \wedge \phi[y/x])$
$\texttt{if}(\pi)\ C_1\ \texttt{else}\ C_2$	$SP(\pi \wedge \phi, C_1) \vee SP(\neg\pi \wedge \phi, C_2)$
$C_1; C_2$	$SP(SP(\phi, C_1), C_2)$,

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 $SP(\phi, x := e) = (\phi \wedge x = e)$. 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

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**

- m > 2 candidates
  n > 2 voters
- L_{QF} linear integer arithmetic
- Threads A = {1, .., n; P}: voters + P is a 'public observer'/ general program
- Program variables $\mathbf{p} = \bigcup_{j=1}^{m}\{c_j\} \cup \bigcup_{i=1}^{n}\bigcup_{j=1}^{m}\bigcup_{k=1}^{3}\{b_{ijk}\}$
- Observable variables $\mathbf{o}_i = \bigcup_{j=1}^{m}\{c_j\} \cup \bigcup_{j=1}^{m}\bigcup_{k=1}^{3}\{b_{ijk}\}$

  $\mathbf{o}_P = \bigcup_{j=1}^{m}\{c_j\}$
- Non-observable variables $\mathbf{n}_i = \mathbf{p} \setminus \mathbf{o}_i$
- Vote Counting (the number of ticks voter i has entered for candidate j) $S_{i,j} \equiv \sum_{k=1}^{3} b_{ijk}$
- **Program C** $c_1 := \sum_{i=1}^{n} S_{i,1}\ ;\ \ldots\ ;\ c_m := \sum_{i=1}^{n} S_{i,m}$
- L_{QF} Presburger arithmetic

Three Ballot Specifications (cont'd)

- Macros to model the protocol

$$S_{i,j} \equiv \sum_{k=1}^{3} b_{ijk}$$

$$B \equiv \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{m} \bigwedge_{k=1}^{3} (b_{ijk} = 0 \vee b_{ijk} = 1)$$

$$V_{i,j} \equiv (S_{i,j} = 2)$$

$$\bar{V}_{i,j} \equiv (S_{i,j} = 1)$$

$$CV_i^{\geq 0} \equiv \bigvee_{j=1}^{m} V_{i,j}$$

$$CV_i^{\leq 1} \equiv \bigwedge_{j=1}^{m} \left(V_{i,j} \Rightarrow \bigwedge_{j'=1, j' \neq j}^{m} \bar{V}_{i,j'} \right)$$

$$CV \equiv \bigwedge_{i=1}^{n} (CV_i^{\geq 0} \wedge CV_i^{\leq 1})$$

$$NU \equiv \bigwedge_{j=1}^{m} \bigvee_{i=1}^{n} V_{i,j}$$

$$NU_{\mathrm{mod}\ i} \equiv \bigwedge_{j=1}^{m} \bigvee_{i'=1, i' \neq i}^{n} V_{i',j}$$

$$I \equiv B \wedge CV \wedge NU$$

$$I_{\mathrm{mod}\ i} \equiv B \wedge CV \wedge NU_{\mathrm{mod}\ i}$$

Voting for at least and at most a candidate

Non-unanimity

Initial states

$$SP(I, C) = I \wedge \left(\mathbf{c} = \left(\sum_{i=1}^{n} S_{i,1}, \ldots, \sum_{i=1}^{n} S_{i,m} \right) \right)$$

$\mathbf{c}$ is the tuple $(c_1, \ldots, c_m)$

Three Ballot Specifications (cont'd)

$\alpha_1 = \neg \mathsf{K}_P V_{1,1}$ the observer P does not know that voter 1 voted for candidate 1

$\alpha_2 = \neg \mathsf{K}_1 V_{2,1}$ voter 1 does not know that voter 2 voted for candidate 1

Vote Privacy Verification

$I \Vdash \Box_C \alpha_1$

$I_{\mathrm{mod}\ 1} \Vdash \Box_C \alpha_2$

$I \nVdash \Box_C \alpha_2$

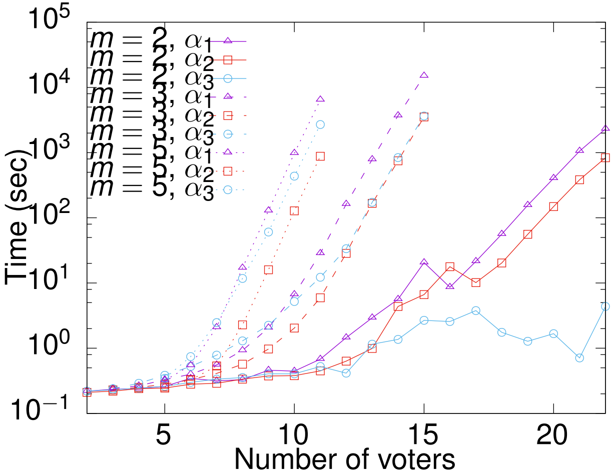
$$SP(I, C) = I \wedge \left(\mathbf{c} = \left(\sum_{i=1}^{n} S_{i,1}, \ldots, \sum_{i=1}^{n} S_{i,m}\right)\right)$$

+

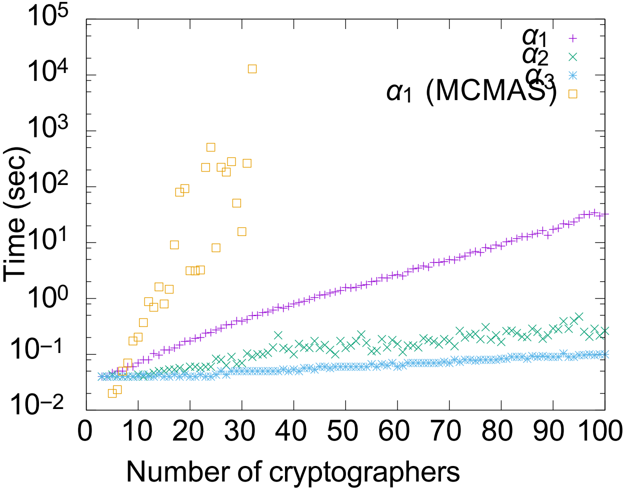
translation of K formulae

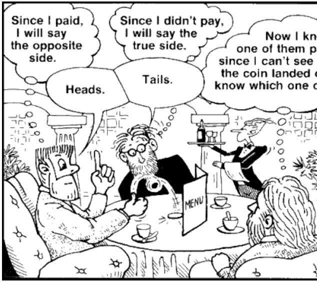
=> Presburger formulas + Z3

Experimental Results (on a simple laptop)



Other Experimental Results





Recall my question re model checking vs SMT solving?

(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.

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 $\Box_C$ 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

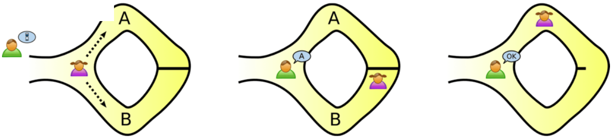
Second Program-Epistemic Language

epistemic K_A operator can appear only after program $\Box_C$ operator…,

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

[FM'23]

- perhaps link the program language and the logic more?
- Announcement logics [Plaza '89] …

Program Syntax

$P ::= \alpha?$	(test/announcement)
$x_G := e$	(assignment)
$\mathbf{new}\, k_G \cdot P$	(declare k visible to G)
$P; Q$	(sequential composition)
$P \sqcap Q$	(nondeterministic choice)

Second Epistemic Logic Syntax $\mathcal{L}_{\mathcal{P}}$

$\alpha ::= \pi$	(atomic predicate)
$\alpha \wedge \alpha'$	(conjunction)
$\neg\alpha$	(negation)
$K_A \alpha$	(knowledge modality)
$[\alpha']\alpha$	(public announcement formula)
$\forall x_{\mathbf{G}} \cdot \alpha$	(universal quantification)

Richer than [IJCAI17]

Let's re-think relational semantics (for the new $\mathcal{L}_{\mathcal{P}}$....)

- ~~$R(v := x; v := 0, \omega) = R(v := 0, \omega)$~~

(wrong if the thread knows the program)

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

Example

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

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

$$wp(v := x, K(x = 0) \vee K(x = 1)) = K(x = 0) \vee K(x = 1)[v \backslash x]$$

True

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

depends on the thread's observability of program execution

Relational Semantics for $\mathcal{L}_{\mathcal{P}}$....

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

For public programs, ...

$$
\begin{aligned}
R_W(P \sqcap Q, s) &= \{s'[c_{Ag} \mapsto l] \mid s' \in R_W(P, s)\} \\
&\quad \cup \{s'[c_{Ag} \mapsto r] \mid s' \in R_W(Q, s)\} \\
R_W(P; Q, s) &= \bigcup_{s' \in R_W(P,s)} \{R_{R^*_W(P,W)}(Q, s')\} \\
R_W(x_G := e, s) &= \{s[k_G \mapsto s(x_G), x_G \mapsto s(e)]\} \\
R_W(\mathbf{new}\, k_G \cdot P, s) &= R^*_W(P, \{s[k_G \mapsto d] \mid d \in \mathrm{D}\}) \\
R_W(\beta?, s) &= \text{if } (W, s) \models \beta \text{ then } \{s\} \text{ else } \varnothing
\end{aligned}
$$

Second, More Expressive Program-Epistemic Language

Richer than [IJCAI17]

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

$$\alpha ::= \pi \mid \alpha \wedge \alpha' \mid \neg\alpha \mid K_{a_i}\alpha \mid [\alpha']\alpha \mid \forall x_{\mathbf{G}} \cdot \alpha \mid \Box_P \alpha$$

- $\Box_P(Kv(secret \mod 2))$
- $K(\Box_P secret \mod 2 = 0)$ ← K in front of program
- $\Box_{DC}\left(K_0\left(x \Leftrightarrow \bigvee_{i=0}^{n-1} p_i\right)\right)$

$$(W,s) \models [\beta]\alpha \quad \text{iff} \quad (W,s) \models \beta \text{ implies } (W_{|\beta}, s) \models \alpha$$

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

$$(W,s) \models \forall x_G \cdot \alpha \quad \text{iff for all } c \in \mathsf{D}, (\bigcup_{d \in \mathsf{D}} \{s'[x_G \mapsto d] \mid s' \in W\}, s[x_G \mapsto c]) \models \alpha$$

Program-based Semantics for $\mathcal{L}_K$....

Linking programs and formula "tighter" than in the first attempt

$wp : \mathcal{L}_P \times \mathcal{L}_K \to \mathcal{L}_K$

$$
\begin{aligned}
wp(P \sqcap Q, \alpha) &= wp(P, \alpha) \wedge wp(Q, \alpha) \\
wp(P; Q, \alpha) &= wp(P, wp(Q, \alpha)) \\
wp(x_G := e, \alpha) &= \forall k_G \cdot [k_G = e](\alpha[x_G \backslash k_G]) \\
wp(\mathbf{new}\, k_G \cdot P, \alpha) &= \forall k_G \cdot wp(P, \alpha) \\
wp(\beta?, \alpha) &= [\beta]\alpha
\end{aligned}
$$

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

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

Main theorem

- $[\![\phi]\!]$ a set of states satisfying FO formula ϕ
- $\alpha \in \mathcal{L}_{PK}$

$$[\![\phi]\!] \models \alpha \Leftrightarrow \text{FO formula } \phi \wedge \neg\tau(\phi, \alpha) \text{ unsatisfiable}$$

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

$$\begin{aligned}
\tau(\phi, \pi) &= \pi \\
\tau(\phi, \neg\alpha) &= \neg\tau(\phi, \alpha) \\
\tau(\phi, \alpha_1 \circ \alpha_2) &= \tau(\phi, \alpha_1) \circ \tau(\phi, \alpha_2) \\
\tau(\phi, \forall x_G \cdot \alpha) &= \forall x_G \cdot \tau(\phi, \alpha)
\end{aligned}$$

$$\begin{aligned}
\tau(\phi, K_a\alpha) &= \forall \mathbf{n} \cdot (\phi \rightarrow \tau(\phi, \alpha)) \\
\tau(\phi, [\beta]\alpha) &= \tau(\phi, \beta) \rightarrow \tau(\phi \wedge \tau(\phi, \beta), \alpha) \\
\tau(\phi, \Box_P\alpha) &= \tau(\phi, wp(P, \alpha))
\end{aligned}$$

One "go" translation for the "full" logic, unlike before

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

Main theorem [FM2023]

- $[\![\phi]\!]$ a set of states satisfying FO formula ϕ
- $\alpha \in \mathcal{L}_{PK}$

$$[\![\phi]\!] \models \alpha \Leftrightarrow \text{FO formula } \phi \wedge \neg\tau(\phi, \alpha) \text{ unsatisfiable}$$

- Mechanised the translation in Haskell

```
tau :: ModalFormula -> Formula a -> ModalFormula
tau phi (Atom p)                = Atom p
tau phi (Neg alpha)             = Neg (tau phi alpha)
tau phi (Conj as)               = Conj [tau phi a | a <- as]
tau phi (Disj as)               = Disj [tau phi a | a <- as]
tau phi (Imp alpha1 alpha2)     = tau phi alpha1 ⇒ tau phi alpha2
tau phi (Equiv alpha1 alpha2) = (tau phi (alpha1 ⇒ alpha2)) ∧ (tau phi (alpha2 ⇒ alpha1))
tau phi (K ag alpha)            = mkForAll (nonobs ag) (phi ⇒ tau phi alpha)
tau phi (Ann beta alpha)        = tau phi beta ⇒ tau (phi ∧ (tau phi beta)) alpha
tau phi (Box p alpha)           = tau phi (wp alpha p)
tau phi (ForAllB n alpha)     = ForAllB n (tau phi alpha)
tau phi (ExistsB n alpha)     = ExistsB n (tau phi alpha)
tau phi (ForAllI n d alpha) = ForAllI n d (tau phi alpha)
tau phi (ExistsI n d alpha) = ExistsI n d (tau phi alpha)
```

$\mathcal{L}_{\mathcal{P}K}$ 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

n	Formula β_1		Formula β_2			Formula β_3		Formula γ	
n	τ_{wp}+Z3	τ_{SP}+Z3	τ_{wp}+CVC5	τ_{wp}+Z3	τ_{SP}+Z3	τ_{wp}+Z3	τ_{SP}+Z3	τ_{wp}+Z3	τ_{SP}+Z3
10	0.05 s	4.86 s	0.01 s	0.01 s	0.01 s	0.01 s	0.01 s	0.01 s	N/A
50	31 s	t.o.	0.41 s	0.05 s	0.06 s	0.03 s	0.02 s	0.03 s	N/A
100	t.o.	t.o.	3.59 s	0.15 s	0.16 s	0.07 s	0.06 s	0.07 s	N/A
200	t.o.	t.o.	41.90 s	1.27 s	0.71 s	0.30 s	0.20 s	0.30 s	N/A

…("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 → 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

Yet Another Program-Epistemic Logics ... *[AAAI2023]*

- 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

$$\alpha ::= \pi \mid \neg\alpha \mid \alpha \wedge \alpha \mid (\mathsf{K}_a \alpha)[\vec{x}/\vec{e}] \mid [\rho]\alpha$$
$$\rho ::= x := e \mid \phi?$$

$$(W, s) \models (K_a\alpha)[\vec{x}/\vec{e}] \textit{ iff for all } s' \in W, \; s' \sim_{\vec{o}_a} s[\vec{x} \mapsto s(\vec{e})] \textit{ implies } (W, s') \models \alpha$$
$$(W, s) \models [\rho]\alpha \quad \textit{iff for all } s' \in R_\rho(s), \; (R_\rho(W), s') \models \alpha$$

We get derived dynamic operators ..

$$[\rho; \rho']\,\alpha \; ::= \; [\rho]\,[\rho']\,\alpha$$
$$[\rho \sqcup \rho']\,\alpha \; ::= \; [\rho]\,\alpha \vee [\rho']\,\alpha$$

Motivation & Aim

Program-Epistemic Logics

Verification Methods of These Logics

Practical Experimentations

Conclusions

Practical Experimentation

	SAT (AAAI 2023)			SAT (IJCAI 2017)			Model Checking (MCMAS)		
Formula	result	time		result	time		result	time	
		$n = 5$	$n = 10$		$n = 5$	$n = 10$		$n = 5$	$n = 10$
$\neg\alpha_1$	unsat	0.07s	70s	unsat	0.03s	0.1s	unsat	0.17s	0.18s
$\neg\alpha_2$	unsat	0.03s	7s	unsat	0.02s	0.1s	unsat	0.10s	0.12s
$\neg\alpha'_2$ ☺	unsat	0.15s	17s	N/A	-	0.1s	unsat	0.20s	0.25s
$\neg\alpha_3$	sat	0.04s	7s	sat	0.01s	0.1s	sat	0.10s	0.12s

Performances on Verifying the Dining-cryptographers Problem

More expressive than IJCAI 2017 --> ***we allow***
$K_a K_b \phi$ and operator K even in front of operator $\Box_C$

Still faster than model checking

Yet Another Program-Epistemic Logics

improvements →

		IJCAI 2017	AAAI 2023	FM 2023
1	K possible before $[prog]$	☹ no	☺ yes	☺ yes
2	only one agent	☹ yes	☺ no	☺ no
3	program public	☹ no	NaN	☺ yes
4	announcements	no	no	yes
5	multiple assignments	☺ yes	☺ yes	☺ no
6	efficiency	x	☹ 2^x	☺ x (due to SSA)

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

• 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