Symbolic Verification of Epistemic Properties in Programs

Ioana Boureanu (Univ. of Surrey, SCCS)

joint work @ IJCAI 2017,
with N. Gorogiannis (Middlesex, Facebook) and F. Raimondi (Middelsex, Amazon)

Asking you...

Motivation & Aim

Program-Epistemic Logic

Verification of Program-Epistemic Logic

Practical Experimentation

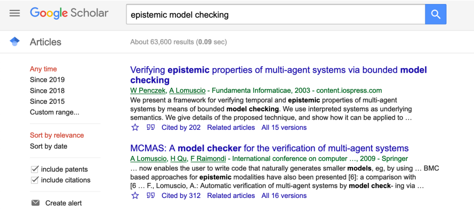
Conclusions

Motivation

- epistemic logics, i.e., logics of knowledge – "knowing logical facts" $\rightarrow$ expressions of rich properties (e.g., unlinkability, anonymity)
- widely used in verification of general-purpose concurrent & distributed SYSTEMS (e.g., Byzantine agreement) via epistemic model checkers such as MCMAS, Verics, MCK, etc....

Motivation

- epistemic logics, i.e., logics of knowledge – “knowing logical facts” $\rightarrow$ expressions of rich properties (e.g., unlinkability, anonymity)
- widely used in verification of general-purpose concurrent & distributed SYSTEMS (e.g., Byzantine agreement) via epistemic model checkers such as MCMAS, Verics, MCK, etc....



Motivation ...

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

  - :(these are not epistemic specifications on program code
  - :(it is hard to capture rich (e.g., first-order) state specifications, since the base logic of most temporal-epistemic verifiers is propositional
  - !!? ... meanwhile, base logics of programs are very expressive + predicate transformers are used to reduce verification to FO queries to SMT solvers ...

Motivation ...

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

  - :(these are not epistemic specifications on program code
  - :(it is hard to capture rich (e.g., first-order) state specifications, since the base logic of most temporal-epistemic verifiers is propositional
  - !!? ... meanwhile, base logics of programs are very expressive + predicate transformers are used to reduce verification to FO queries to SMT solvers ...

Motivation ...

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

  - :(these are not epistemic specifications on program code
  - :(it is hard to capture rich (e.g., first-order) state specifications, since the base logic of most temporal-epistemic verifiers is propositional
  - !!? ... meanwhile, base logics of programs are very expressive + predicate transformers are used to reduce verification to FO queries to SMT solvers ...

Motivation ...

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

  - :(these are not epistemic specifications on program code
  - :(it is hard to capture rich (e.g., first-order) state specifications, since the base logic of most temporal-epistemic verifiers is propositional
  - !!? ... meanwhile, base logics of programs are very expressive + predicate transformers are used to reduce verification to FO queries to SMT solvers ...

Motivation ...

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

  - :(these are not epistemic specifications on program code
  - :(it is hard to capture rich (e.g., first-order) state specifications, since the base logic of most temporal-epistemic verifiers is propositional
  - !!? ... meanwhile, base logics of programs are very expressive + predicate transformers are used to reduce verification to FO queries to SMT solvers ...

Aim

- be able to verify epistemic properties of programs
- agents can OBSERVE certain program variables
- the program (i.e., state-transition relation) is KNOWN to all agents
- focus on S5-*like epistemic properties about program states*

"*agent observer1 knows that variable x is equal to $y + 5$*"

"*agent observer2 does not know that variable x is equal to $y + 5$*"

Aim

- be able to verify epistemic properties of programs
- agents can OBSERVE certain program variables
- the program (i.e., state-transition relation) is KNOWN to all agents
- focus on S5-*like epistemic properties about program states*

"*agent observer1 knows that variable x is equal to $y + 5$*"

"*agent observer2 does not know that variable x is equal to $y + 5$*"

Aim

- be able to verify epistemic properties of programs
- agents can OBSERVE certain program variables
- the program (i.e., state-transition relation) is KNOWN to all agents
- focus on S5-*like epistemic properties about program states*

"*agent observer1 knows that variable x is equal to $y + 5$*"

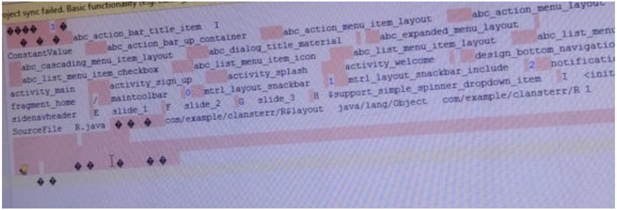
"*agent observer2 does not know that variable x is equal to $y + 5$*"

Aim

- be able to verify epistemic properties of programs
- agents can OBSERVE certain program variables
- the program (i.e., state-transition relation) is KNOWN to all agents
- focus on S5-*like* *epistemic properties about program states*

  “*agent observer1 knows that variable x is equal to* $y + 5$”

  “*agent observer2 does not know that variable x is equal to* $y + 5$”



Motivation & Aim

Program-Epistemic Logic

Verification of Program-Epistemic Logic

Practical Experimentation

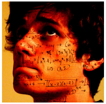
Conclusions

Setup

- $\mathcal{A}$ — a finite set of *agents* 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 agent $A \in \mathcal{A}$ can *observe*
- $\mathbf{n}_A = \mathbf{p} \setminus \mathbf{o}_A$ — variables agent $A \in \mathcal{A}$ *cannot observe*

Syntax

Epistemic Language $\mathcal{L}_{\mathrm{K}}$



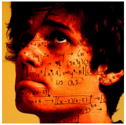
- $\mathcal{L}_{QF}$ *base language* = a quantifier-free, FO language
- $\mathcal{L}_{FO}$ extension of $\mathcal{L}_{\mathrm{QF}}$ with quantifiers

$$\phi ::= \pi \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \phi_1 \Rightarrow \phi_2 \mid \forall x.\, \phi \mid \exists x.\, \phi$$

- $\mathcal{L}_{\mathrm{K}}$ extension of $\mathcal{L}_{\mathrm{QF}}$ with epistemic modalities K_A

$$\alpha ::= \pi \mid \neg\alpha \mid \alpha_1 \wedge \alpha_2 \mid \alpha_1 \vee \alpha_2 \mid \alpha_1 \Rightarrow \alpha_2 \mid \mathrm{K}_A\alpha$$

Program-Epistemic Specifications $\mathcal{L}_{\Box K}$



- $\mathcal{C}$ a (possibly infinite) set of *commands*
- $\mathcal{L}_{\Box K}$ extends $\mathcal{L}_{K}$ with every formula $\beta = \Box_{C}\alpha$, meaning "*at all final states of C, α holds*"

Example

"at the end of the vote-counting, a partial observer (who can see certain aspects of the program) does not know that voter 1 vote for candidate 1":

$$\Box_{EVotingProgram} \neg K_{public-observer} V_{1,1},$$

where $V_{1,1}$ is a formula in $\mathcal{L}_{QF}$ which here is linear integer arithmetic.

First-order Semantics

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

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

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

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

Interpretation of a program specification β

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

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

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 $[\![\phi]\!], s \Vdash \beta$.

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

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

Motivation & Aim

Program-Epistemic Logic

Verification of Program-Epistemic Logic

Practical Experimentation

Conclusions

Reducing to First-Order Validity

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

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

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).

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

$$\begin{array}{llll} \tau(\phi, \pi) & = \pi & \tau(\phi, \alpha_1 \circ \alpha_2) & = \tau(\phi, \alpha_1) \circ \tau(\phi, \alpha_2) \\ \tau(\phi, \neg\alpha) & = \neg\tau(\phi, \alpha) & \tau(\phi, \mathrm{K}_A\alpha) & = \forall \mathbf{n}_A.\, (\phi \Rightarrow \tau(\phi, \alpha)) \end{array}$$

Over-approximation

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

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

- a function $f : \mathcal{L}_{\mathrm{FO}} \times \mathcal{C} \to \mathcal{L}_{\mathrm{FO}}$ *over-approximates the strongest postcondition* iff ... $[\![f(\phi, C)]\!] \supseteq R_C([\![\phi]\!])$ for all $\phi \in \mathcal{L}_{\mathrm{FO}}$ and $C \in \mathcal{C}$

When the strongest postcondition can only be over-approximated (such as in programming languages with unbounded loops), we show that the validity of *positive* epistemic specifications reduces to that of first-order fragments, in a sound but incomplete way.

Motivation & Aim

Program-Epistemic Logic

Verification of Program-Epistemic Logic

Practical Experimentation

Conclusions

Simple, Loop-Free Programming Language

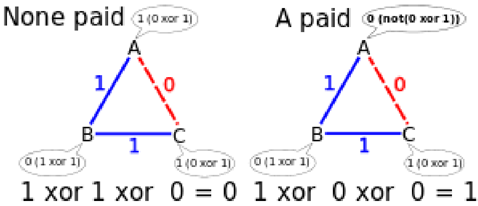
Command C	$SP(\phi, C)$
$x := *$	$\exists y.\, \phi[y/x]$
$x := e$	$\exists y.\, (x = e[y/x] \wedge \phi[y/x])$
$\texttt{if}(\pi)\ C_1\ \texttt{else}\ C_2$	$SP(\pi \wedge \phi, C_1) \vee SP(\neg\pi \wedge \phi, C_2)$
$C_1 ; C_2$	$SP(SP(\phi, C_1), C_2)$,

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 $\|\phi\|$.

An Example – The Dining Cryptographers

– used as evaluation case-study in verifying epistemic properties



– dinner may have been paid by their employer, or by one of the agents.
– reveal whether one of the agents paid, but without revealing which one.
– each pair of adjacent agents sees a coin
– each announces the result of XORing three Booleans: the two coins observable by her and the status of whether she paid for the dinner.

– the XOR of all announcements is proven to be equal to the disjunction of whether any agent paid.

Instantiation

agents $\mathcal{A} = \{0, \ldots, n-1\}$

program variables $\mathbf{p} = \{x\} \cup \{p_i, c_i \mid 0 \leq i < n\}$,

x is the XOR of announcements; p_i encodes whether agent i has paid; and, c_i encodes the coin shared between agents $i-1$ and i.

observable variables by $i \in \mathcal{A}$ $\mathbf{o}_i = \{x, p_i, c_i, c_{i+1 \bmod n}\}$,
$\mathbf{n}_i = \mathbf{p} \setminus \mathbf{o}_i$.

protocol = an assignment C:

$$x := \bigoplus_{i=0}^{n-1} p_i \oplus c_i \oplus c_{(i+1 \bmod n)} \qquad (C)$$

initial states, I == at most one agent paid

$$I = \bigwedge_{i=0}^{n-1} \left(p_i \Rightarrow \bigwedge_{j=0, j \neq i}^{n-1} \neg p_j\right)$$

strongest postcondition

$$SP(I, C) = I \wedge \left(x \Leftrightarrow \bigoplus_{i=0}^{n-1} p_i \oplus c_i \oplus c_{(i+1 \bmod n)}\right)$$

Specifications

$\alpha_1 = \neg p_0 \Rightarrow \left(\left(\mathsf{K}_0 \bigwedge_{i=0}^{n-1} \neg p_i\right) \vee \left(\bigwedge_{i=1}^{n-1} \neg \mathsf{K}_0 p_i\right)\right)$
if agent 0 has not paid then she knows that no agent paid, or (in case an agent paid) she does not know which one.

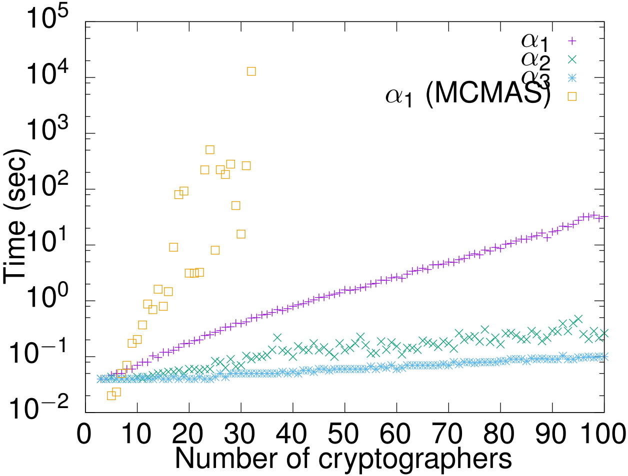
$\alpha_2 = \mathsf{K}_0 \left(x \Leftrightarrow \bigvee_{i=0}^{n-1} p_i\right)$
agent 0 knows that x is true iff one of the agents paid.

$\alpha_3 = \mathsf{K}_0 p_1$
agent 0 knows that agent 1 has paid

To verify $I \Vdash \Box_C \alpha_1$, $I \Vdash \Box_C \alpha_2$ and $I \nVdash \Box_C \alpha_3$

We construct the QBF formula $SP(I, C) \wedge \neg\tau(SP(I, C), \alpha_i)$, feed it to Z3, and test for unsatisfiability, as per our results.

Experimental Results

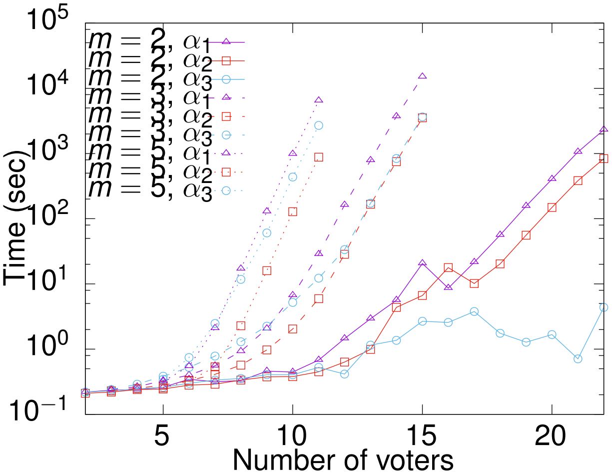


(i) MCMAS is faster, or equally fast, for $n \leq 7$, but slower for all $n > 7$;
(ii) we can be faster than MCMAS by a factor of > 100 (e.g., when $n = 32$) when checking α_1, whilst when verifying α_3 our speed-up is of several orders of magnitudes.

exp. specs.: a 4-core 2.4 GHz Intel Core i7 MacBook Pro with 16 GB of RAM running OS X 10.11.6. The version of MCMAS is 1.2.2 and Z3 is 4.5.1; both tools have been compiled from source on the target machine.

More ...

- a more complicated example on the ThreeBallot voting protocol (e.g., $\mathcal{L}_{\mathrm{FO}}$ moved from QBFs to Presburger arithmetics.)



Motivation & Aim

Program-Epistemic Logic

Verification of Program-Epistemic Logic

Practical Experimentation

Conclusions

Take-home Message

- we gave *program-epistemic* specifications, expressing requiremenst 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 tools such as SMT solvers
- we traded off temporal expressivity, to deal with arbitrary programming languages
- space for improvements... in temporal operators, common knowledge, translations modulo bespoke semantics...

Take-home Message

- we gave *program-epistemic* specifications, expressing requiremenst 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 tools such as SMT solvers
- we traded off temporal expressivity, to deal with arbitrary programming languages
- space for improvements... in temporal operators, common knowledge, translations modulo bespoke semantics...

Take-home Message

- we gave *program-epistemic* specifications, expressing requiremenst 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 tools such as SMT solvers
- we traded off temporal expressivity, to deal with arbitrary programming languages
- space for improvements... in temporal operators, common knowledge, translations modulo bespoke semantics...

Take-home Message

- we gave *program-epistemic* specifications, expressing requiremenst 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 tools such as SMT solvers
- we traded off temporal expressivity, to deal with arbitrary programming languages
- space for improvements... in temporal operators, common knowledge, translations modulo bespoke semantics...

Thank you

... for listening....

i.boureanu@surrey.ac.uk

Cheeky Slide...

- Do you know a British national who wishes to do a PhD in formal verification of privacy(GBP 22k/year stipend, NCSC project, with BT and the 5G Innovation Centre)? https://www.jobs.ac.uk/job/BTV392/phd-studentship-opportunity-security-analysis-o

- Do you know a prospective postdoc in formal verification of privacy (EPSRC 3-year project, with Thales and Vector)? https://www.jobs.ac.uk/job/BTX925/research-fellow-in-formal-verification-of-priva