Model Checking Multi-Agent Systems

Agents, Systems & Algorithms

Francesco Belardinelli

Laboratoire IBISC

October 16, 2017
Model Checking Multi-agent Systems

- A Case Study: the Bit Transmission Problem
- Agents
- Multi-agent Systems
- The Specification Language CTLK
- Model Checking MAS against CTLK
- MCMAS

This presentation is based on [FHMV95].
A sender $S$ wants to communicate the value of a bit to a receiver $R$, by using an unreliable communication channel.

The channel may drop messages, but cannot tamper with messages.

One mechanism to achieve communication is as follows:

- $S$ immediately starts sending the bit to $R$, and continues to do so until it receives an acknowledgement from $R$.
- $R$ does nothing until it receives the bit; from then on, it sends messages to $S$ acknowledging the receipt.
- $S$ stops sending the bit to $R$ when she receives the first acknowledgement from $R$, and the protocol terminates here.
Each agents $a$ is supposed to

- be in some *local state* $l_a$ modelling the information the agent possesses about the systems
Each agents $a$ is supposed to

- be in some *local state* $l_a$ modelling the information the agent possesses about the systems
- perform some *actions* $\alpha_a, \alpha'_a, \ldots$
Each agent $a$ is supposed to

- be in some *local state* $l_a$ modelling the information the agent possesses about the systems
- perform some *actions* $\alpha_a, \alpha'_a, \ldots$
- ... according to a *protocol* $Pr_a$, which returns the set $Pr_a(l_a)$ of actions enabled in each local state $l_a$. 

**Definition 1 (Agent [FHMV95])**

An agent is a tuple $a = \langle L_a, \text{Act}_a, Pr_a \rangle$ where

- $L_a$ is the set of local states $l_a$
- $\text{Act}_a$ is the set of actions $\alpha_a, \alpha'_a$
- $Pr_a : L_a \to \text{Act}_a$ is the protocol function

“A special” agent $e$ is usually included in the system description to account for the behaviour of the environment.
Each agents $a$ is supposed to

- be in some *local state* $l_a$ modelling the information the agent possesses about the systems
- perform some *actions* $\alpha_a, \alpha'_a, \ldots$
- ...according to a *protocol* $Pr_a$, which returns the set $Pr_a(l_a)$ of actions enabled in each local state $l_a$.

**Definition 1 (Agent [FHMV95])**

An *agent* is a tuple $a = \langle L_a, Act_a, Pr_a \rangle$ where

- $L_a$ is the set of local states $l_a, l'_a$
- $Act_a$ is the set of actions $\alpha_a, \alpha'_a$
- $Pr_a : L_a \rightarrow Act_a$ is the protocol function

A “special” agent $e$ is usually included in the system description to account for the behaviour of the environment.
Example 2 (BTP)
The sender is the agent $S = \langle L_S, Act_S, Pr_S \rangle$ where

- $L_S$ contains local states 0, 1, (0, $ack$), (1, $ack$)
- $Act_S$ contains actions $sendbit$, $skip$
- $Pr_S : L_S \rightarrow Act_S$ is the protocol function defined as
  - $Pr_S(0) = Pr_S(1) = \{sendbit\}$
  - $Pr_S((0, ack)) = Pr_S((1, ack)) = \{skip\}$

The receiver is the agent $R = \langle L_R, Act_R, Pr_R \rangle$ where

- $L_R$ contains local states $\lambda$, 0, 1
- $Act_R$ contains actions $skip$, $sendack$
- $Pr_R : L_R \rightarrow Act_R$ is the protocol function defined as
  - $Pr_R(\lambda) = \{skip\}$
  - $Pr_R(0) = Pr_R(1) = \{sendack\}$

The environment is the agent $e = \langle L_e, Act_e, Pr_e \rangle$ where

- $L_e$ contains local states $S$, $R$, $SR$, none
- $Act_e$ contains actions $S$, $R$, $SR$, none
- $Pr_e : L_e \rightarrow Act_e$ is the protocol function defined as
  - $Pr_e(S) = Pr_e(R) = Pr_e(SR) = Pr_e(\text{none}) = Act_e$
The state of the system is the composition of the local states of all agents.

**Definition 3 (Global State)**

Fix a set $Ag = \{e, a_1, \ldots, a_n\}$ of agents. A *global state* is a tuple $s = \langle l_e, l_1, \ldots, l_n \rangle \in G$ ($G$ being set of all global states).
The state of the system is the composition of the local states of all agents.

**Definition 3 (Global State)**

Fix a set $Ag = \{e, a_1, \ldots, a_n\}$ of agents. A *global state* is a tuple $s = \langle l_e, l_1, \ldots, l_n \rangle \in G$ ($G$ being set of all global states).

A multi-agent system (MAS)
- describes the temporal evolution of a *group* $Ag$ of agents
The state of the system is the composition of the local states of all agents.

**Definition 3 (Global State)**
Fix a set $Ag = \{e, a_1, \ldots, a_n\}$ of agents.
A *global state* is a tuple $s = ⟨l_e, l_1, \ldots, l_n⟩ \in G$ ($G$ being set of all global states).

A multi-agent system (MAS)
- describes the temporal evolution of a *group Ag of agents*
- from some *initial global state* $s_0$ ...
The state of the system is the composition of the local states of all agents.

**Definition 3 (Global State)**

Fix a set $Ag = \{e, a_1, \ldots, a_n\}$ of agents.

A *global state* is a tuple $s = \langle l_e, l_1, \ldots, l_n \rangle \in G$ ($G$ being set of all global states).

A multi-agent system (MAS)

- describes the temporal evolution of a group $Ag$ of agents
- from some *initial global state* $s_0$ . . .
- . . . according to the *transition function* $\tau$, which returns the set $\tau(s, \alpha)$ of successor states for each current global state $s \in G$ and *enabled* joint action $\alpha = \langle \alpha_e, \alpha_1, \ldots, \alpha_n \rangle \in Act_e \times Act_1 \times \cdots \times Act_n$. 

Multi-agent Systems
Interaction

The state of the system is the composition of the local states of all agents.

**Definition 3 (Global State)**

Fix a set $Ag = \{e, a_1, \ldots, a_n\}$ of agents.
A *global state* is a tuple $s = \langle l_e, l_1, \ldots, l_n \rangle \in G$ ($G$ being set of all global states).

A multi-agent system (MAS)

- describes the temporal evolution of a *group* $Ag$ of agents
- from some *initial global state* $s_0$ ...
- ... according to the *transition function* $\tau$, which returns the set $\tau(s, \alpha)$ of successor states for each current global state $s \in G$ and *enabled* joint action $\alpha = \langle \alpha_e, \alpha_1, \ldots, \alpha_n \rangle \in Act_e \times Act_1 \times \cdots \times Act_n$.

**Definition 4 (MAS [FHMV95])**

A *multi-agent system* is a tuple $M = \langle Ag, I, \tau \rangle$ where

- $Ag = \{e, a_1, \ldots, a_n\}$ is the set of agents
- $I$ is the set of initial global states $s_0, s'_0, \ldots$
- $\tau: G \times Act \to 2^G$ is the transition function such that
  - $\tau(s, \alpha)$ is defined iff for all agents $a \in Ag$, $\alpha_a \in Pr_a(l_a)$. 
Example 5 (BTP)

The MAS corresponding to the BTP is the tuple $\mathcal{M}_{BTP} = \langle Ag, I, \tau \rangle$ where

- $Ag = \{e, S, R\}$ is the set of agents
- $I$ contains global states $s_{0e} = \langle 0, \lambda, l_e \rangle$ and $s'_{0e} = \langle 1, \lambda, l_e \rangle$, for any $l_e \in L_e$
- the transition function $\tau$ can be given as
  - $\tau(\langle 0, \lambda, l_e \rangle, \langle \text{sendbit}, \text{skip}, \alpha_e \rangle) = \langle 0, \lambda, \alpha_e \rangle$ for $l_e \in \{R, \text{none}\}$
  - $\tau(\langle 0, \lambda, l_e \rangle, \langle \text{sendbit}, \text{skip}, \alpha_e \rangle) = \langle 0, 0, \alpha_e \rangle$ for $l_e \in \{S, SR\}$
  - ...

- The transition relation $s \rightarrow s'$ holds iff for some joint action $\alpha$, $s' \in \tau(s, \alpha)$.
  - $Post(s) = \{ s' \in G \mid s \rightarrow s' \}$
  - $Post^+(s)$ is the transitive closure of $Post(s)$

- A run $r$ is an infinite sequence $s^0 \rightarrow s^1 \rightarrow s^2 \rightarrow \ldots$
  - for $i \geq 0$, $r(i) = s^i$
  - $s' \in Post^+(s)$ iff for some run $r$ from $s$, $s' = r(i)$ for some $i \geq 0$
  - the set $S$ of reachable states is defined as $\bigcup_{s_0 \in I} Post^+(s_0)$
To express properties of MAS we introduce

- a set $AP$ of atomic propositions $p_0, p_1, \ldots$
- an interpretation function $\pi : G \rightarrow 2^{AP}$

Atomic propositions are interpreted on global states!

**Definition 6 (IS [FHMV95])**
Given a MAS $\mathcal{M}$ and an interpretation $\pi$ for a set $AP$ of atomic propositions, an interpreted (multi-agent) system is a tuple $\mathcal{P} = \langle \mathcal{M}, \pi \rangle$.

**Example 7 (BTP)**
Consider set $AP = \{bit = 0, bit = 1, recbit, recack, sentbit, sentack\}$ and interpretation $\pi$ such that

- $bit = k \in \pi(s)$ iff $l_S = k$
- $recbit \in \pi(s)$ iff $l_R = 0$ or $l_R = 1$
- $recack \in \pi(s)$ iff $l_S = (0, ack)$ or $l_S = (1, ack)$
- $sentbit \in \pi(s)$ iff $l_e = S$ or $l_e = SR$
- $sentack \in \pi(s)$ iff $l_e = R$ or $l_e = SR$
The Specification Language CTL

Temporal Logic

- **Safety properties:**
  - “the value of the bit is always defined”

- **Liveness properties:**
  - “the bit is eventually received”

- **Conditional liveness:**
  - “every bit sent is eventually received”
The Specification Language CTL
Temporal Logic

- **Safety properties:**
  - “the value of the bit is always defined”
- **Liveness properties:**
  - “the bit is eventually received”
- **Conditional liveness:**
  - “every bit sent is eventually received”

We start with the Computation-tree Temporal Logic CTL defined by the following BNF

\[ \phi ::= p | \neg \phi | \phi \rightarrow \phi | \forall X \phi | \forall \Box \phi | \forall \lozenge \phi | \forall \phi U \phi | \exists X \phi | \exists \Box \phi | \exists \lozenge \phi | \exists \phi U \phi \]

“the value of the bit is always defined” \(\equiv \forall \Box (bit = 0 \lor bit = 1)\)

“the bit is eventually received” \(\equiv \forall \lozenge \text{recbit} \quad \text{(or } \exists \lozenge \text{recbit}?\text{)}\)

“every bit sent is eventually received” \(\equiv \forall \Box (sendbit \rightarrow \forall \lozenge \text{recbit})\)
The Specification Language
Epistemic Logic

- Knowledge:
  - “the sender knows the value of the bit”
- Knowledge of temporal facts:
  - “the sender knows that the receiver will receive the bit”
- Temporal evolution of knowledge
  - “the receiver will eventually know the value of the bit”
- Knowledge of mixed facts
  - “the sender knows that the receiver will eventually know the value of the bit”
The Specification Language
Epistemic Logic

- Knowledge:
  - “the sender knows the value of the bit”
- Knowledge of temporal facts:
  - “the sender knows that the receiver will receive the bit”
- Temporal evolution of knowledge
  - “the receiver will eventually know the value of the bit”
- Knowledge of mixed facts
  - “the sender knows that the receiver will eventually know the value of the bit”

Consider the Epistemic Temporal Logic CTLK defined by the following BNF

\[
\phi ::= p | \neg \phi | \phi \rightarrow \phi | \forall X \phi | \forall \Diamond \phi | A \phi U \phi | \exists X \phi | \exists \Box \phi | \exists \Diamond \phi | E \phi U \phi | K_a \phi | D_A \phi | C_A \phi
\]

“the sender knows the value of the bit” \( \equiv K_S \; \text{bit} = 0 \lor K_S \; \text{bit} = 1 \)

“the sender knows that the receiver will receive the bit” 
\( \equiv K_S \; \forall \Diamond \; \text{recbit} \)

“the receiver will eventually know the value of the bit” 
\( \equiv \forall \Diamond (K_R \; \text{bit} = 0 \lor K_R \; \text{bit} = 1) \)

“the sender knows that the receiver will eventually know the value of the bit” 
\( \equiv K_S \; \forall \Diamond (K_S \; \text{bit} = 0 \lor K_S \; \text{bit} = 1) \)
“Knowledge” ::= truth in all indistinguishable states (for agent $a$).

Two global states $s$ and $s'$ are *indistinguishable* for agent $a$ iff the information possessed by $a$ is the same in $s$ and $s'$.

... iff $l_a = l'_a$

We introduce an indistinguishability relation $\sim_a$ such that $s \sim_a s'$ iff $l_a = l'_a$.

$\sim_a$ is an equivalence relation (in line with epistemic logic S5)

$[s]_a$ is the equivalence class of $s$ according to $\sim_a$

Distributed knowledge (in a group $A$ of agents) corresponds to what any wise man would know: $\bigcap_{a \in A} \sim_a$

“the sender and receiver have distributed knowledge of the value of the bit” $\equiv D_{S,R} bit = 0 \lor D_{S,R} bit = 1$

Common knowledge (in a group $A$ of agents) corresponds to what any fool would know: $(\bigcup_{a \in A} \sim_a)^+$

“after the acknowledgement is delivered, the sender and receiver have common knowledge of the value of the bit” $\equiv \forall \Box (\text{recack} \rightarrow (C_{S,R} bit = 0 \lor C_{S,R} bit = 1))$
The Specification Language CTLK

Knowledge

- $p = \text{“il fait beau à Evry”}$
- $q = \text{“il fait beau à Paris”}$
The Specification Language CTLK

Knowledge

- $p = "il fait beau à Evry"
- $q = "il fait beau à Paris"
- $K_{Alice}p = "Alice sait qu’il fait beau à Evry"
- $\neg K_{Alice}q = "mais elle ne sait pas s’il fait beau à Paris"
- $K_{Alice}\neg K_{Alice}q = "Alice sait qu’elle ne sait pas s’il fait beau à Paris"
Definition 8 (Semantics)

We define whether an IS $\mathcal{P}$ satisfies a formula $\phi$ at state $s$ as follows:

$(\mathcal{P}, s) \models p$ iff $p \in \pi(s)$

$(\mathcal{P}, s) \models \neg \phi$ iff $(\mathcal{P}, s) \not\models \phi$

$(\mathcal{P}, s) \models \phi \rightarrow \phi'$ iff $(\mathcal{P}, s) \not\models \phi$ or $(\mathcal{P}, s) \models \phi'$

$(\mathcal{P}, s) \models \forall X \phi$ iff for all runs $r$ from $s$, $(\mathcal{P}, r(1)) \models \phi$

$(\mathcal{P}, s) \models \forall \phi U \phi'$ iff for all runs $r$ from $s$, for some $i \geq 0$, $(\mathcal{P}, r(i)) \models \phi'$ and

for all $0 \leq j < i$, $(\mathcal{P}, r(j)) \models \phi$

$(\mathcal{P}, s) \models \exists X \phi$ iff for some run $r$ from $s$, $(\mathcal{P}, r(1)) \models \phi$

$(\mathcal{P}, s) \models \exists \phi U \phi'$ iff for some run $r$ from $s$, for some $i \geq 0$, $(\mathcal{P}, r(i)) \models \phi'$ and

for all $0 \leq j < i$, $(\mathcal{P}, r(j)) \models \phi$

$(\mathcal{P}, s) \models K_a \phi$ iff for all states $s' \in S$, if $s' \sim_a s$ then $(\mathcal{P}, s') \models \phi$

$(\mathcal{P}, s) \models D_A \phi$ iff for all states $s' \in S$, if $s' \sim_a s$ for all $a \in A$, then $(\mathcal{P}, s') \models \phi$

$(\mathcal{P}, s) \models C_A \phi$ iff for all states $s' \in S$, if $s' \sim_A s$ then $(\mathcal{P}, s') \models \phi$

where $\sim_A = (\bigcup_{a \in A} \sim_a)^+$
Derived clauses:

\[(\mathcal{P}, s) \models \forall \square \phi \quad \text{iff for all runs } r \text{ from } s, \text{ for all } i \geq 0, (\mathcal{P}, r(i)) \models \phi\]

\[(\mathcal{P}, s) \models \forall \lozenge \phi \quad \text{iff for all runs } r \text{ from } s, \text{ for some } i \geq 0, (\mathcal{P}, r(i)) \models \phi\]

\[(\mathcal{P}, s) \models \exists \square \phi \quad \text{iff for some run } r \text{ from } s, \text{ for all } i \geq 0, (\mathcal{P}, r(i)) \models \phi\]

\[(\mathcal{P}, s) \models \exists \lozenge \phi \quad \text{iff for some run } r \text{ from } s, \text{ for some } i \geq 0, (\mathcal{P}, r(i)) \models \phi\]

A formula \(\phi\) is *true* in an IS \(\mathcal{P}\) iff for all \(s_0 \in I\), \((\mathcal{P}, s_0) \models \phi\).
Do the specifications above hold for the Bit Transmission Problem?

Definition 9 (Model Checking Problem)
Given an IS $\mathcal{P}$ and a CTLK formula $\phi$, is it the case that $\mathcal{P} \models \phi$?

Theorem 10 ([CGP99, LR06])

Model checking IS against CTLK is PTIME-complete.

Instead of checking whether $\mathcal{P} \models \phi$, we equivalently compute the satisfaction set $[\phi] = \{s \in \mathcal{P} \mid (\mathcal{P}, s) \models \phi\}$ and check whether $I \subseteq [\phi]$.

We first rewrite formulas in the complete base $\{\neg, \wedge, \forall \diamond, \exists U, \exists X, D_A, C_A\}$
We make use of a labelling algorithm that label each state \( s \) with the subformulas \( \psi \) of \( \phi \) that are satisfied in \( s \), starting with atomic propositions and working towards \( \phi \) (we assume that all immediate subformulas of \( \psi \) have already been labelled.)

**Input** an IS \( \mathcal{P} \) and a CTLK formula \( \phi \)

**Output** the set \([\phi]\) of states that satisfy \( \phi \)

- label each state \( s \) with the atomic propositions in \( \pi(s) \)
- for each subformula \( \psi \) of \( \phi \)
  - \( \neg\psi \): label \( s \) with \( \neg\psi \) if \( s \) is not already labelled with \( \psi \)
  - \( \psi \land \psi' \): label \( s \) with \( \psi \land \psi' \) if \( s \) is already labelled with \( \psi \) and \( \psi' \)
  - \( \forall \diamond \psi \):
    - if \( s \) is labelled with \( \psi \), then label \( s \) with \( \forall \diamond \psi \)
    - repeat: label any state with \( \forall \diamond \psi \) if all successors are labelled with \( \forall \diamond \psi \), until no change.
  - \( \exists\psi U\psi' \):
    - if \( s \) is labelled with \( \psi' \), then label \( s \) with \( E\psi U\psi' \)
    - repeat: label any state with \( E\psi U\psi' \) if it is labelled with \( \psi \) and at least one of its successors is labelled with \( E\psi U\psi' \), until no change.
  - \( \exists X\psi \):
    - label \( s \) with \( EX\psi \) if at least one of its successors is labelled with \( \psi \).
  - \( DA\psi \):
    - label \( s \) with \( DA\psi \) if all states indistinguishable for all \( a \in A \) are labelled with \( \psi \).
  - \( CA\psi \):
    - if \( s \) is labelled with \( \psi \), then label \( s \) with \( CA\psi \)
    - repeat: label any state with \( CA\psi \) if for all agents \( a \in A \), all states indistinguishable for \( a \) are labelled with \( CA\psi \), until no change.
Algorithm 1 Computation of the satisfaction set $[\phi]$

Require: an IS $P$ and a CTLK formula $\phi$
Ensure: the set $[\phi]$ of states that satisfy $\phi$

1: switch $(\phi)$:
2: case $\bot$:
3: return $\emptyset$;
4: case $p$:
5: return $\pi(p)$;
6: case $\neg \psi$:
7: return $S \setminus [\psi]$;
8: case $\psi \land \psi'$:
9: return $[\psi] \cap [\psi']$;
10: case $\forall \diamond \psi$:
11: return $\text{SAT}_{\forall \diamond} (\phi)$;
12: case $\exists \psi U \psi'$:
13: return $\text{SAT}_{\exists U} (\phi, \phi')$;
14: case $\exists X \psi$:
15: return $\text{SAT}_{\exists X} (\phi)$;
16: case $D_A \psi$:
17: return $\text{SAT}_D (\phi, A)$;
18: case $K_a \psi$:
19: return $\text{SAT}_D (\phi, \{a\})$;
20: case $C_A \psi$:
21: return $\text{SAT}_C (\phi, A)$;
The auxiliary functions $\text{SAT}_\exists X$ and $\text{SAT}_\forall \diamond$

\[
\text{pre}_\exists (Y) = \{ s \in S \mid \text{for some } s', s \xrightarrow{} s' \text{ and } s' \in Y \} \\
\text{pre}_\forall (Y) = \{ s \in S \mid \text{for all } s', s \xrightarrow{} s' \text{ implies } s' \in Y \}
\]

**Algorithm 2** Computation of function $\text{SAT}_\exists X (\phi)$

1: $X := \text{SAT}(\phi)$
2: $Y := \text{pre}_\exists (X)$
3: return $Y$

**Algorithm 3** Computation of function $\text{SAT}_\forall \diamond (\phi)$

1: $X := S$
2: $Y := \text{SAT}(\phi)$
3: repeat
4: $X := Y$
5: $Y := Y \cup \text{pre}_\forall (Y)$
6: until $X = Y$
7: return $Y$
Algorithm 4 Computation of function $SAT_{\exists U}(\phi, \phi')$

1: $W := SAT(\phi)$
2: $X := S$
3: $Y := SAT(\phi')$
4: repeat
5: $X := Y$
6: $Y := Y \cup (W \cap pre_\exists Y)$
7: until $X = Y$
8: return $Y$
Model Checking Multi-agent Systems
The Auxiliary Functions $SAT_D$ and $SAT_C$

\[ pre_D(Y, A) = \{ s \in S \mid \bigcap_{a \in A} [s]_a \subseteq Y \} \]

**Algorithm 5** Computation of function $SAT_D(\phi, A)$

1: $X := SAT(\phi)$
2: $Y := pre_D(X, A)$
3: return $Y$

**Algorithm 6** Computation of function $SAT_C(\phi, A)$

1: $X := \emptyset$
2: $Y := SAT(\phi)$
3: repeat
4: $X := Y$
5: $Y := Y \cup \bigcap_{a \in A} pre_D(Y, \{a\})$
6: until $X = Y$
7: return $Y$
Theorem 11
The labelling algorithm is sound and complete.

- **Soundness**: if state $s$ is labelled with formula $\phi$, then $s \models \phi$.
- **Completeness**: if $s \models \phi$, then state $s$ is labelled with formula $\phi$.

Theorem 12
The complexity of model checking IS against CTLK has a worst case of $O(|\phi| \times (|S| \times E))$. 
Beyond Explicit Representation: Symbolic Model Checking

- State explosion problem: adding a boolean variable doubles the size of the model
  - explicit model checking: \( \approx 10^6 \) states

- (Ordered) Binary Decision Diagrams [BCM+92]: compact representation of boolean functions (sets of states, relations)
  - allow for more efficient manipulation
  - symbolic model checking: \( \approx 10^{30} \) states
  - supported in MCMAS

- Wealth of techniques to perform model checking
  - bounded model checking
  - symmetry reduction
  - predicate abstraction
  - . . .
Model Checking Multi-agent Systems

- A Case Study: the Bit Transmission Problem
- Agents
- Multi-agent Systems
- The Specification Language CTLK
- Model Checking MAS against CTLK
- MCMAS
Symbolic Model Checking: $10^{20}$ States and Beyond.

E. M. Clarke, O. Grumberg, and D. A. Peled.
*Model Checking*.

*Reasoning About Knowledge*.

A. Lomuscio and F. Raimondi.
The complexity of model checking concurrent programs against CTLK specifications.