Temporal logics for multi-agent systems

Nicolas Markey
LSV, CNRS & ENS Cachan, France

Journées Nationales
Lyon, 21-22 January 2013
Verification of computerised systems

- Computers are everywhere
Verification of computerised systems

- Computers are everywhere

- Bugs are everywhere...

Toyota to recall Prius hybrids over ABS software

By Martyn Williams

February 1, 2010 02:16 AM ET

IDG News Service - Toyota plans to recall around 400,000 of its Prius hybrid cars to replace software that controls the anti-lock braking system (ABS), the auto maker said Tuesday.
Verification of computerised systems

- Computers are everywhere

- Bugs are everywhere...

- Verification should be everywhere!
Model checking and synthesis

system:

[Diagram of system with tanks, pump, and action symbols]

property:

[Diagram of property with X mark on tanks]

$\text{model-checking algorithm}$

$AG(\neg B.\text{overfull} \land \neg B.\text{dried\_up})$

yes/no
Model checking and synthesis

system:

[http://www.embedded.com]

property:

\[ \text{synthesis algorithm} \]

\[ \text{AG}(\neg B.\text{overfull} \land \neg B.\text{dried\_up}) \]
Outline of the presentation

1. Introduction
   - formal verification
   - model checking and synthesis

2. Classical temporal logics: CTL and LTL
   - expressing properties of “closed” systems

3. Temporal logics for games: ATL and extensions
   - expressing properties of interacting systems
   - extensions to non-zero-sum games

4. Conclusions and future works
Outline of the presentation

1. Introduction
   - formal verification
     model checking and synthesis

2. Classical temporal logics: CTL and LTL
   - expressing properties of “closed” systems

3. Temporal logics for games: ATL and extensions
   - expressing properties of interacting systems
     extensions to non-zero-sum games

4. Conclusions and future works
CTL and LTL: temporal logics for closed systems

- **atomic propositions:** $\bigcirc$, $\bigcirc$, ...

- **boolean combinators:** $\neg \varphi$, $\varphi \lor \psi$, $\varphi \land \psi$, ...

- **path quantifiers:** $E \varphi$, $\forall \varphi$, $\exists \varphi$

- **temporal modalities:** $X \varphi$, "next $\varphi$", $\varphi U \psi$, "$\varphi$ until $\psi$", $\varphi U \top$, $\neg F \neg \varphi$, $G \varphi$, "always $\varphi$"
CTL and LTL: temporal logics for closed systems

- atomic propositions: \( \bigcirc, \Box, ... \)
- boolean combinators: \( \neg \varphi, \varphi \lor \psi, \varphi \land \psi, \ldots \)
- path quantifiers:

\[
E \varphi \\
A \varphi
\]
CTL and LTL: temporal logics for closed systems

- **atomic propositions**: \( \bigcirc, \bigcirc, \ldots \)
- **boolean combinators**: \( \neg \varphi, \varphi \lor \psi, \varphi \land \psi, \ldots \)
- **path quantifiers**:

  \[
  \text{E}\varphi \quad \text{A}\varphi
  \]

- **temporal modalities**:

  \[
  \text{X}\varphi \quad \varphi \text{ U } \psi
  \]

  "next \( \varphi \)"

  "\( \varphi \) until \( \psi \)"
CTL and LTL: temporal logics for closed systems

- **atomic propositions:** ⊤, ⬤, ...
- **boolean combinators:** ¬φ, φ ∨ ψ, φ ∧ ψ, ...
- **path quantifiers:**
  - Eφ
  - Aφ
- **temporal modalities:**
  - Xφ
  - φ U ψ
  - true U φ ≡ Fφ
  - ¬ F ¬ φ ≡ Gφ
  - “next φ”
  - “φ until ψ”
  - “eventually φ”
  - “always φ”
CTL and LTL: temporal logics for closed systems

- CTL: each temporal modality is in the immediate scope of a path quantifier.
CTL and LTL: temporal logics for closed systems

- CTL: each temporal modality is in the immediate scope of a path quantifier.

\[ E F \text{ is reachable} \]
CTL and LTL: temporal logics for closed systems

- CTL: each temporal modality is in the immediate scope of a path quantifier.

\[ E F \quad \text{is reachable} \]

Diagram:

- Initial state with a path quantifier.
- States connected by transitions.
CTL and LTL: temporal logics for closed systems

- CTL: each temporal modality is in the immediate scope of a path quantifier.

$$\mathbf{E} \mathbf{G} (\mathbf{E} \mathbf{F} \bigcirc)$$ there is a path along which $\bigcirc$ is always reachable.
CTL and LTL: temporal logics for closed systems

- CTL: each temporal modality is in the immediate scope of a path quantifier.

\[ \text{EG}(\text{EF} \ p) \]  
there is a path along which \( p \) is always reachable.
CTL and LTL: temporal logics for closed systems

- CTL: each temporal modality is in the immediate scope of a path quantifier.

\[
\mathbf{E} \mathbf{G}(\mathbf{E} \mathbf{F} \circled{p}) \quad \text{there is a path along which } \circled{p} \text{ is always reachable}
\]
CTL and LTL: temporal logics for closed systems

- **CTL**: each temporal modality is in the immediate scope of a path quantifier.

\[ \neg E(\neg \bullet) \cup \bullet \quad \text{in order to reach } \bullet, \text{ we have to visit } \bullet \]
CTL and LTL: temporal logics for closed systems

- CTL: each temporal modality is in the immediate scope of a path quantifier.

\[ \neg E(\neg \bigcirc) U \bigcirc \] in order to reach \( \bigcirc \), we have to visit \( \bigcirc \)
CTL and LTL: temporal logics for closed systems

- CTL: each temporal modality is in the immediate scope of a path quantifier.

**Theorem ([CE81, QS82])**

*CTL model checking is PTIME-complete.*

*CTL symbolic model checking is PSPACE-complete.*
CTL and LTL: temporal logics for closed systems

- CTL: each temporal modality is in the immediate scope of a path quantifier.

**Theorem ([CE81,QS82])**

*CTL model checking is PTIME-complete.*

*CTL symbolic model checking is PSPACE-complete.*

- LTL: $E\varphi$ or $A\varphi$, where $\varphi$ has no path quantifier.
CTL and LTL: temporal logics for closed systems

- CTL: each temporal modality is in the immediate scope of a path quantifier.

**Theorem ([CE81,QS82])**

*CTL model checking is PTIME-complete.*

*CTL symbolic model checking is PSPACE-complete.*

- LTL: $E\varphi$ or $A\varphi$, where $\varphi$ has no path quantifier.

  $E(G F \bigcirc)$ there is a path visiting $\bigcirc$ infinitely many times
CTL and LTL: temporal logics for closed systems

- **CTL**: each temporal modality is in the immediate scope of a path quantifier.

**Theorem ([CE81,QS82])**

*CTL model checking is PTIME-complete.*

*CTL symbolic model checking is PSPACE-complete.*

- **LTL**: $E\varphi$ or $A\varphi$, where $\varphi$ has no path quantifier.

$$E(G F \bigcirc) \quad \text{there is a path visiting } \bigcirc \text{ infinitely many times}$$
CTL and LTL: temporal logics for closed systems

- CTL: each temporal modality is in the immediate scope of a path quantifier.

**Theorem ([CE81,QS82])**

*CTL model checking is PTIME-complete.*

*CTL symbolic model checking is PSPACE-complete.*

- LTL: $E \varphi$ or $A \varphi$, where $\varphi$ has no path quantifier.

\[
A[(F \bullet) \Rightarrow (F G \neg \bullet)]
\]

any path that visits \( \bullet \) visits \( \circ \) finitely many times
**CTL and LTL: temporal logics for closed systems**

- **CTL:** each temporal modality is in the immediate scope of a path quantifier.

**Theorem ([CE81,QS82])**

*CTL model checking is PTIME-complete.*

*CTL symbolic model checking is PSPACE-complete.*

- **LTL:** \(E\varphi\) or \(A\varphi\), where \(\varphi\) has no path quantifier.

\[
A[(F \bigcirc) \Rightarrow (F G \neg \bigcirc)]
\]

any path that visits \(\bigcirc\) visits \(\bigcirc\) finitely many times
CTL and LTL: temporal logics for closed systems

- CTL: each temporal modality is in the immediate scope of a path quantifier.

**Theorem ([CE81,QS82])**

*CTL model checking is PTIME-complete.*

*CTL symbolic model checking is PSPACE-complete.*

- LTL: $E\phi$ or $A\phi$, where $\phi$ has no path quantifier.

**Theorem ([SC85])**

*LTL (symbolic) model checking is PSPACE-complete.*
Outline of the presentation

1. Introduction
   - formal verification
     model checking and synthesis

2. Classical temporal logics: CTL and LTL
   - expressing properties of “closed” systems

3. Temporal logics for games: ATL and extensions
   - expressing properties of interacting systems
     extensions to non-zero-sum games

4. Conclusions and future works
Reasoning about interacting systems

Concurrent games

A concurrent game is made of

- a transition system;
Reasoning about interacting systems

**Concurrent games**

A concurrent game is made of

- a transition system;
- a set of agents (or players);

![Diagram of a concurrent game with states q0, q1, q2 and transitions between them.]

player 1  
player 2
Reasoning about interacting systems

**Concurrent games**

A concurrent game is made of

- a transition system;
- a set of agents (or players);
- a table indicating the transition to be taken given the actions of the players.

![Diagram](image)
Reasoning about interacting systems

Concurrent games
A concurrent game is made of
- a transition system;
- a set of agents (or players);
- a table indicating the transition to be taken given the actions of the players.

Turn-based games
A turn-based game is a game where only one agent plays at a time.
Reasoning about open systems

**Strategies**

A strategy for a given player is a function telling what to play depending on what has happened previously.
A strategy for a given player is a function telling what to play depending on what has happened previously.

Strategy for player □:
alternately go to ● and ●.
Reasoning about open systems

Strategies

A strategy for a given player is a function telling what to play depending on what has happened previously.

Strategy for player □:
alternately go to □ and □.
Reasoning about open systems

**Strategies**

A *strategy* for a given player is a function telling what to play depending on what has happened previously.

Strategy for player □:
alternately go to • and •.
Reasoning about open systems

Strategies

A strategy for a given player is a function telling what to play depending on what has happened previously.

Strategy for player □:
alternately go to 🟡 and 🟢.

[Diagram showing a strategy tree with nodes and arrows connecting them, indicating the possible actions and transitions.]
Reasoning about open systems

**Strategies**

A *strategy* for a given player is a function telling what to play depending on what has happened previously.

Strategy for player □:
alternately go to ● and ◼.
A strategy for a given player is a function telling what to play depending on what has happened previously.

Strategy for player \( \square \): alternately go to \( \bigcirc \) and \( \bigcirc \).
Strategies

A strategy for a given player is a function telling what to play depending on what has happened previously.

Strategy for player \( \square \): alternately go to \( \bigcirc \) and \( \bigcirc \).
Reasoning about open systems

**Strategies**

A strategy for a given player is a function telling what to play depending on what has happened previously.

Strategy for player □:
alternately go to ○ and ○.

![Diagram](image-url)
Reasoning about open systems

**Strategies**

A strategy for a given player is a function telling what to play depending on what has happened previously.

Strategy for player □:
alternately go to ○ and ○.
Alternating-time Temporal Logic

ATL formulas are built inductively using atomic propositions, Boolean combinations, temporal modalities $X$ and $U$, and strategy quantifiers:

$\langle A \rangle \varphi$ expresses that $A$ has a strategy to enforce $\varphi$. 
Alternating-time Temporal Logic

ATL formulas are built inductively using atomic propositions, Boolean combinations, temporal modalities $X$ and $U$, and strategy quantifiers:

$\langle A \rangle \varphi$ expresses that $A$ has a strategy to enforce $\varphi$. 

\[ \vphantom{\text{ATL}} \]
Alternating-time Temporal Logic

ATL formulas are built inductively using atomic propositions, Boolean combinations, temporal modalities $X$ and $U$, and strategy quantifiers:

$$\langle\langle A \rangle\rangle \varphi$$ expresses that $A$ has a strategy to enforce $\varphi$.
Alternating-time Temporal Logic

ATL formulas are built inductively using atomic propositions, Boolean combinations, temporal modalities $\textbf{X}$ and $\textbf{U}$, and strategy quantifiers:

$\langle\langle A \rangle\rangle \varphi$ expresses that $A$ has a strategy to enforce $\varphi$. 

![Diagram of ATL formulas with nodes and edges illustrating the logic's structure.]
Alternating-time Temporal Logic

ATL formulas are built inductively using atomic propositions, Boolean combinations, temporal modalities $X$ and $U$, and strategy quantifiers:

$$\langle A \rangle \varphi$$ expresses that $A$ has a strategy to enforce $\varphi$. 
**Alternating-time Temporal Logic**

ATL formulas are built inductively using atomic propositions, Boolean combinations, temporal modalities $\mathbf{X}$ and $\mathbf{U}$, and strategy quantifiers:

\[
\langle\langle A \rangle\rangle \varphi \text{ expresses that } A \text{ has a strategy to enforce } \varphi.
\]
Alternating-time Temporal Logic

ATL formulas are built inductively using atomic propositions, Boolean combinations, temporal modalities $X$ and $U$, and strategy quantifiers:

$$\langle A \rangle \varphi \text{ expresses that } A \text{ has a strategy to enforce } \varphi.$$
Alternating-time Temporal Logic

ATL formulas are built inductively using atomic propositions, Boolean combinations, temporal modalities $X$ and $U$, and strategy quantifiers:

$$\langle A \rangle \varphi$$ expresses that $A$ has a strategy to enforce $\varphi$. 

$$\langle \square \rangle F \equiv \langle \Diamond \rangle G(p)$$
Alternating-time Temporal Logic

ATL formulas are built inductively using atomic propositions, Boolean combinations, temporal modalities $\mathbf{X}$ and $\mathbf{U}$, and strategy quantifiers:

$s A \rangle \varphi$ expresses that $A$ has a strategy to enforce $\varphi$.

**Theorem ([AHK02,HLW06])**

ATL model checking is PTIME-complete.
ATL symbolic model checking is EXPTIME-complete.
Another semantics: ATL with strategy contexts [BDLM09]

Consider the following strategy of Player: “always go to”. In the remaining tree, Player can always enforce a visit to F.
Another semantics: ATL with strategy contexts [BDLM09]

Consider the following strategy of Player blue: “always go to yellow”;

$$\langle \square \rangle G(\langle \Box \rangle F \bigcirc)$$
Another semantics: ATL with strategy contexts [BDLM09]

\[ \langle \Box \rangle G(\langle \Box \rangle F \circ) \]

• consider the following strategy of Player \( \circ \): “always go to \( \Box \)”;
Another semantics: ATL with strategy contexts [BDLM09]

\[
\langle \bigcirc \rangle \ \mathbf{G} \langle \bigboxdot \rangle \ \mathbf{F} \ \bigcirc
\]

consider the following strategy of Player \( \bigcirc \): “always go to \( \bigboxdot \);”

in the remaining tree, Player \( \bigboxdot \) can always enforce a visit to \( \bigcirc \).
ATL with strategy contexts

Definition

$\text{ATL}_{sc}$ has two new strategy quantifiers: $\langle \cdot A \rangle \varphi$ and $\parallel A \parallel \varphi$.

- $\langle \cdot A \rangle$ is similar to $\langle A \rangle$ but assigns the corresponding strategy to $A$ for evaluating $\varphi$;
- $\parallel A \parallel$ drops the assigned strategies for $A$. 

Theorem

$\text{ATL}_{sc}$ is strictly more expressive than $\text{ATL}$. 
ATL with strategy contexts

Theorem

$ATL_{sc}$ is strictly more expressive than ATL.
**Theorem**

$\text{ATL}_{sc}$ is strictly more expressive than ATL.

**Proof**

\[
\langle A \rangle \varphi \equiv \langle \text{Agt} \rangle \langle A \rangle \hat{\varphi}
\]
ATL with strategy contexts

**Theorem**

$ATL_{sc}$ is strictly more expressive than ATL.

**Proof**

$\langle 1\cdot \rangle (\langle 2\cdot \rangle Xa \land \langle 2\cdot \rangle Xb)$ is only true in the second game. But ATL cannot distinguish between these two games.
What $\text{ATL}_{sc}$ can express

- All $\text{ATL}^*$ properties:
What $\text{ATL}_{sc}$ can express

- All $\text{ATL}^*$ properties:
- Client-server interactions for accessing a shared resource:

$$
\langle \text{Server} \rangle \ G \left[ \bigwedge_{c \in \text{Clients}} \langle \cdot \cdot \rangle F \ \text{access}_c \right.
\left. \wedge \neg \bigwedge_{c \neq c'} \text{access}_c \wedge \text{access}_{c'} \right]
$$
What $\text{ATL}_{sc}$ can express

- All $\text{ATL}^{*}$ properties:
- Client-server interactions for accessing a shared resource:
  \[
  \langle \cdot \text{Server} \cdot \rangle \text{ G } \left[ \bigwedge_{c \in \text{Clients}} \langle \cdot c \cdot \rangle \text{ F access}_c \right] \land \left( \neg \bigwedge_{c \neq c'} \text{access}_c \land \text{access}_{c'} \right)
  \]

- Existence of Nash equilibria:
  \[
  \langle \cdot A_1, \ldots, A_n \cdot \rangle \bigwedge_i \left( \langle \cdot A_i \cdot \rangle \varphi_{A_i} \Rightarrow \varphi_{A_i} \right)
  \]
What $\text{ATL}_{sc}$ can express

- All $\text{ATL}^*$ properties:
- Client-server interactions for accessing a shared resource:
  \[
  \langle \text{Server} \rangle \ G \ \land \ \begin{cases} 
  
  \bigwedge_{c \in \text{Clients}} \langle \cdot \rangle F \text{ access}_c \\
  
  \neg \bigwedge_{c \neq c'} \text{ access}_c \land \text{ access}_{c'} 
  \end{cases}
  \]

- Existence of Nash equilibria:
  \[
  \langle \cdot A_1, \ldots, A_n \rangle \ \land \ (\langle \cdot A_i \rangle \ \varphi_{A_i} \ \Rightarrow \ \varphi_{A_i})
  \]

- Existence of dominating strategy:
  \[
  \langle \cdot A \rangle \ [B] (\neg \varphi \ \Rightarrow \ [A] \neg \varphi)
  \]
Theorem

Given a CGS $\mathcal{C}$, a state $\ell_0$ and an ATL$_{sc}$ formula $\varphi$, we can build an alternating parity tree automaton $A$ s.t.

$$\mathcal{L}(A) \neq \emptyset \iff \mathcal{C}, \ell_0 \models \emptyset \varphi.$$ 

$A$ has size $d$-exponential, where $d$ is the maximal number of nested quantifiers.

Proof (Theorem [DLM10]) Model checking ATL$_{sc}$ formulas with $d$ strategy quantifiers is $d$-EXPTIME-complete.
Model checking \( \text{ATL}_{sc} \)

**Theorem**

Given a CGS \( C \), a state \( l_0 \) and an \( \text{ATL}_{sc} \) formula \( \varphi \), we can build an alternating parity tree automaton \( A \) s.t.

\[
\mathcal{L}(A) \neq \emptyset \iff C, l_0 \models \emptyset \varphi.
\]

\( A \) has size \( d \)-exponential, where \( d \) is the maximal number of nested quantifiers.

**Proof**

Theorem ([DLM10])

Model checking \( \text{ATL}_{sc} \) formulas with \( d \) strategy quantifiers is \( d \)-\text{EXPTIME}-complete.
Related formalisms

Strategy logic [CHP07,MMV10]

- first-order quantification over strategies;
- strategies are assigned to players;

Example

\[ \langle A \rangle \ G(\langle B \rangle \ F \bigcirc) \] would be written

\[ \exists \sigma_A. \ \forall \sigma_B. \ (A \ plays \ \sigma_A \ \land \ B \ plays \ \sigma_B) \ \Rightarrow \ G(\exists \sigma'_B. \ B \ plays \ \sigma'_B \ \Rightarrow \ F \bigcirc) \]

Other related formalisms

- Quantified Decision \( \mu \)-calculus [Pin07];
- Stochastic Game Logic [BBG+07];
- ATL with irrevocable strategies [ÅGJ08], ...
Conclusions

Temporal logics for games

- ATL and ATL\textsubscript{sc} are convenient formalisms for reasoning about interacting systems;
- ATL\textsubscript{sc} is much more expressive: equilibria, client-server interactions... Well-suited for multi-agent systems;
- There is a price for this expressiveness: high complexity of the model-checking algorithm.

http://www.cassting-project.eu
Conclusions

Temporal logics for games

- ATL and ATL$_{sc}$ are convenient formalisms for reasoning about interacting systems;
- ATL$_{sc}$ is much more expressive: equilibria, client-server interactions... Well-suited for multi-agent systems;
- There is a price for this expressiveness: high complexity of the model-checking algorithm.

Future works

- fragments of ATL$_{sc}$ with better complexity;
- more realistic setting: stochastic strategies, partial observation, bounded memory...
- effective synthesis of strategies.

http://www.cassting-project.eu
Model checking ATL_{sc} – Algorithm

Tree-automata approach

The unwinding tree is accepted by a deterministic tree automaton;
Model checking $\text{ATL}_{sc}$ – Algorithm

Tree-automata approach

- The unwinding tree is accepted by a deterministic tree automaton;
A strategy is encoded as a labelling of the unwinding tree;
We can mark outcomes corresponding to selected strategies;
We mark the tree with extra propositions $p_l$ and $p_r$, and require that it satisfies $\mathbf{A}(\mathbf{G} p_o \Rightarrow p_l \mathbf{U} p_r)$;
We require that subtrees rooted at a $p_l$ or $p_r$ node is accepted by the automaton for $\varphi$ or $\varphi'$, respectively;
We can build a tree automaton accepting all trees that *can be labelled* with correct strategies. This requires turning the alternating tree automaton into a non-deterministic one, which yields an *exponential-size* automaton.
Hardness

QLTL extends LTL with quantification over atomic propositions:

Example

\[ \forall a. \exists b. \ G (b \iff X a) \]
Hardness

QLTL extends LTL with quantification over atomic propositions:

Example

$$\forall a. \exists b. \ G(b \leftrightarrow X a)$$

Theorem (SVW87)

Satisfiability of a QLTL formula is $k$-EXPSPACE-complete, where $k$ is the alternation-depth of the formula.
Hardness

QLTL extends LTL with quantification over atomic propositions:

Example

$$\forall a. \exists b. \quad G(b \Leftrightarrow X a)$$
Hardness

QLTL extends LTL with quantification over atomic propositions:

Example

\[ \forall a. \exists b. \ G(b \equiv X a) \]

Theorem (SVW87)

Satisfiability of a QLTL formula is \(k\)-EXPSPACE-complete, where \(k\) is the alternation-depth of the formula.
Theorem

ATL model checking is \( k \)-EXPSPACE-hard for formulas with alternation depth \( k \).
Hardness

Example

\[ [A] \langle B \rangle \quad \text{G} \quad \Rightarrow \quad \text{G}(\langle Z \rangle \quad X \quad X \quad b) \quad \iff \quad X \quad (\langle Z \rangle \quad X \quad X \quad a) \]

Theorem

\( ATL_{sc} \) model checking is \( k\)-\( EXPSPACE \)-hard for formulas with alternation depth \( k \).