Temporal logics for multi-agent systems
Expressiveness and algorithms

François Laroussinie
LIAFA – Univ. Paris-Diderot
Nicolas Markey
LSV – ENS Cachan

ESSLLI 2015
Barcelona, 10-14 august 2015
In order to compute the current date, the Zune player has to compute the current year, given the total number of days elapsed since January 1st, 1980. For instance, if 777 days elapsed since January 1st, 1980, then 366 days elapsed in 1980; 365 days elapsed in 1981; 46 days elapsed since January 1st, 1982.
Verification of computerized systems – The Zune bug

In order to compute the current date, the Zune player has to compute the current year, given the total number of days elapsed since January 1st, 1980.
Verification of computerized systems – The Zune bug

In order to compute the current date, the Zune player has to compute the current year, given the total number of days elapsed since January 1st, 1980.

For instance, if 777 days elapsed since January 1st, 1980, then

- 366 days elapsed in 1980;
- 365 days elapsed in 1981;
- 46 days elapsed since January 1st, 1982.
function current_year(int days)

year = 1980;

while (days > 365) {
    if (IsLeapYear(year)) {
        if (days > 366) {
            days -= 366;
            year += 1;
        }
    } else {
        days -= 365;
        year += 1;
    }
}

return(year);
Verification of computerized systems – The Zune bug

```java
function current_year(int days) {
  year = 1980;

  while (days > 365) {
    if (IsLeapYear(year)) {
      if (days > 366) {
        days -= 366;
        year += 1;
      }
    } else {
      days -= 365;
      year += 1;
    }
  }

  return(year);
}
```

Example

on Feb. 15th, 1982:

<table>
<thead>
<tr>
<th>days</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>777</td>
<td>1980</td>
</tr>
<tr>
<td>411</td>
<td>1981</td>
</tr>
<tr>
<td>46</td>
<td>1982</td>
</tr>
<tr>
<td>46</td>
<td>1982</td>
</tr>
</tbody>
</table>
function current_year(int days)
year = 1980;

while (days > 365) {
    if (IsLeapYear(year)) {
        if (days > 366) {
            days -= 366;
            year += 1;
        }
    } else {
        days -= 365;
        year += 1;
    }
}
return(year);

Example
on Feb. 15th, 1982:

days = 777
year = 1980

days = 411
year = 1981
function current_year(int days)

year = 1980;

while (days > 365) {
    if (IsLeapYear(year)) {
        if (days > 366) {
            days -= 366;
            year += 1;
        }
    } else {
        days -= 365;
        year += 1;
    }
}

return(year);

Example

on Feb. 15th, 1982:

\[
\begin{align*}
\text{days} &= 777 \\
\text{year} &= 1980 \\
\text{days} &= 411 \\
\text{year} &= 1981 \\
\text{days} &= 46 \\
\text{year} &= 1982
\end{align*}
\]
Verification of computerized systems – The Zune bug

```plaintext
function current_year(int days)
    year = 1980;

    while (days > 365) {
        if (IsLeapYear(year)) {
            if (days > 366) {
                days -= 366;
                year += 1;
            }
        } else {
            days -= 365;
            year += 1;
        }
    }

    return(year);
```

Example

on Feb. 15th, 1982:

- days = 777
  - year = 1980
- days = 411
  - year = 1981
- days = 46
  - year = 1982
function current_year(int days)
year = 1980;

while (days > 365) {
    if (IsLeapYear(year)) {
        if (days > 366) {
            days -= 366;
            year += 1;
        }
    } else {
        days -= 365;
        year += 1;
    }
}

return(year);

Example
on Dec. 31st, 2008:

<table>
<thead>
<tr>
<th>days</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>9132</td>
<td>1980</td>
</tr>
</tbody>
</table>

days = 366  
year = 2008

days = 366  
year = 2008
Verification of computerized systems – The Zune bug

```c
function current_year(int days) {
    year = 1980;
    while (days > 365) {
        if (IsLeapYear(year)) {
            if (days > 366) {
                days -= 366;
                year += 1;
            }
        } else {
            days -= 365;
            year += 1;
        }
    }
    return(year);
}
```

Example

on Dec. 31st, 2008:

- days = 9132
- year = 1980

- days = 366
- year = 2008
Verification of computerized systems – The Zune bug

```c
function current_year(int days)
{
    year = 1980;

    while (days > 365) {
        if (IsLeapYear(year)) {
            if (days > 366) {
                days -= 366;
                year += 1;
            }
        } else {
            days -= 365;
            year += 1;
        }
    }
    return(year);
}
```

Example

on Dec. 31st, 2008:

- days = 9132
- year = 1980
- days = 366
- year = 2008

- days = 366
- year = 2008
function current_year(int days)
    year = 1980;

    while (days > 365) {
        if (IsLeapYear(year)) {
            if (days > 366) {
                days -= 366;
                year += 1;
            }
        } else {
            days -= 365;
            year += 1;
        }
    }

    return(year);

Example
on Dec. 31st, 2008:
- days = 9132
  year = 1980
- days = 366
  year = 2008
- days = 366
  year = 2008
Formal verification

- proofs of correctness;
- exhaustive methods;
- automated techniques.

Theorem (Turing, 1936)
The halting problem for Turing machines is undecidable.

Different techniques (model-based) testing, statistical verification, theorem proving, model checking...
Formal verification

- proofs of correctness;
- exhaustive methods;
- automated techniques.

Theorem (Turing, 1936)

The halting problem for Turing machines is undecidable.
Formal verification

- proofs of correctness;
- exhaustive methods;
- automated techniques.

Theorem (Turing, 1936)

The halting problem for Turing machines is undecidable.

Different techniques

- (model-based) testing, statistical verification
Formal verification

- proofs of correctness;
- exhaustive methods;
- automated techniques.

Theorem (Turing, 1936)

The halting problem for Turing machines is undecidable.

Different techniques

- (model-based) testing,
- statistical verification
- theorem proving
Formal verification

- proofs of correctness;
- exhaustive methods;
- automated techniques.

Theorem (Turing, 1936)

The halting problem for Turing machines is undecidable.

Different techniques

- (model-based) testing,
- statistical verification
- theorem proving
- model checking
- ...
Model checking and synthesis

system:

property

\[ A \bigwedge \neg B.\text{overfull} \wedge \neg B.\text{dried\_up} \]

model-checking algorithm

yes/no

[http://www.embedded.com]
Model checking and synthesis

system:

property

[http://www.embedded.com]
Outline of the presentation

1 Introduction

2 Basics of temporal logics
   - convenient formalism for expressing properties of reactive systems
   - efficient verification algorithms for finite-state models

3 Temporal logics for multi-agent systems
   - reasoning about interacting components in complex systems
   - temporal logics for expressing controllability properties
Outline of the presentation

1. Introduction

2. Basics of temporal logics
   - convenient formalism for expressing properties of reactive systems
   - efficient verification algorithms for finite-state models

3. Temporal logics for multi-agent systems
   - reasoning about interacting components in complex systems
   - temporal logics for expressing controllability properties
Modelling reactive systems: finite-state transition systems

We consider finite-state machines for modelling reactive systems.
We consider finite-state machines for modelling reactive systems.

Example (An ATM)

- **Idle**: State where the ATM is ready to receive a card.
- **Return card**: State where the ATM waits for a card to be inserted.
- **Check**: State where the ATM checks the card.
- **Pin**: State where the ATM waits for a pin to be entered.
- **Wrong pin**: State where the ATM checks if the pin is correct.
- **Amount**: State where the ATM waits for an amount to be entered.
- **Give cash**: State where the ATM dispenses cash.
- **Check balance**: State where the ATM checks the balance.
- **Positive**: State where the ATM checks if the balance is positive.
- **Negative**: State where the ATM checks if the balance is negative.
- **Stop**: State where the ATM stops processing.

Transitions:
- Insert card to Return card.
- Unknown card from Return card.
- Card ok from Check to Pin.
- Retry from Wrong pin.
- Pin ok from Pin.
- Amount from Pin.
- Negative from amount.
- Positive from check balance.
Finite-state machines can be defined as product of smaller models.
Modelling reactive systems: finite-state transition systems

Finite-state machines can be defined as product of smaller models.

Example (An elevator)

- floor_0
- floor_1
- floor_2
- serv_0
- serv_1
- serv_2
- cabin
Finite-state machines can be defined as product of smaller models.

Example (An elevator)
Modelling reactive systems: finite-state transition systems

Finite-state machines can be defined as product of smaller models.

Example (An elevator)

- **floor**: Represents the different floors in the elevator.
- **serv**: Represents the elevator service.
- **cl**: Represents the car location.
- **op**: Represents the operation of the elevator.
- **idle**: Represents the idle state of the elevator.
- **req**: Represents the request for the elevator.
- **cabin**: Represents the cabin doors.
- **doors**: Represents the doors of the elevator.
- **buttons**: Represents the buttons for the elevator.
Modelling reactive systems: finite-state transition systems

Formally:

**Definition**

A finite-state machine (a.k.a. Kripke structure) is a 4-tuple $\mathcal{M} = \langle Q, q_0, R, \ell \rangle$ where

- $Q$ is a finite (or countable) set of **states**;
- $q_0 \in Q$ is the initial state;
- $R \subseteq Q \times Q$ is a **transition relation** satisfying
  \[ \forall q \in Q. \exists q' \in Q. (q, q') \in R \]
- $\ell : Q \rightarrow 2^{AP}$ **labels states** with atomic propositions.
Finite-state machines are used to represent infinite behaviours.
Modelling reactive systems: finite-state transition systems

Finite-state machines are used to represent infinite behaviours.

Example (Execution tree of a finite-state machine)
Modelling reactive systems: finite-state transition systems

Finite-state machines are used to represent infinite behaviours.

Example (Execution tree of a finite-state machine)
Modelling reactive systems: finite-state transition systems

Formally:

**Definition**

- A **finite run** of $\mathcal{M}$ from $q$ is a pair $(w, \ell)$ where
  - $w = (q_i)_{0 \leq i \leq n}$ is a finite sequence of states with
    - $q_0 = q$
    - $(q_i, q_{i+1}) \in R$ for all $0 \leq i \leq n - 1$;
  - $\ell$ labels states with atomic propositions;
Modelling reactive systems: finite-state transition systems

Formally:

**Definition**

- A **finite run** of $\mathcal{M}$ from $q$ is a pair $(w, \ell)$ where
  - $w = (q_i)_{0 \leq i \leq n}$ is a finite sequence of states with
    - $q_0 = q$
    - $(q_i, q_{i+1}) \in R$ for all $0 \leq i \leq n - 1$;
  - $\ell$ labels states with atomic propositions;

\[ i \rightarrow c \rightarrow p \rightarrow w \rightarrow p \rightarrow a \]
Modelling reactive systems: finite-state transition systems

Formally:

**Definition**

- A **finite run** of $\mathcal{M}$ from $q$ is a pair $(w, \ell)$ where
  - $w = (q_i)_{0 \leq i \leq n}$ is a finite sequence of states with
    - $q_0 = q$
    - $(q_i, q_{i+1}) \in R$ for all $0 \leq i \leq n - 1$;
  - $\ell$ labels states with atomic propositions;

![Diagram showing a finite run](image-url)
Modelling reactive systems: finite-state transition systems

Formally:

**Definition**

- A **finite run** of $\mathcal{M}$ from $q$ is a pair $(w, \ell)$ where
  - $w = (q_i)_{0 \leq i \leq n}$ is a finite sequence of states with
    - $q_0 = q$
    - $(q_i, q_{i+1}) \in R$ for all $0 \leq i \leq n - 1$;
  - $\ell$ labels states with atomic propositions;

- The **execution tree** of $\mathcal{M}$ from $q$ is a pair $(T, \ell)$ where
  - $T$ is the set of finite runs of $\mathcal{M}$ from $q$;
  - $\ell$ labels runs with the label of their last state.
Modelling reactive systems: finite-state transition systems

Formally:

**Definition**

- A **finite run** of $\mathcal{M}$ from $q$ is a pair $(w, \ell)$ where
  - $w = (q_i)_{0 \leq i \leq n}$ is a finite sequence of states with
    - $q_0 = q$
    - $(q_i, q_{i+1}) \in R$ for all $0 \leq i \leq n - 1$;
  - $\ell$ labels states with atomic propositions;

- The **execution tree** of $\mathcal{M}$ from $q$ is a pair $(T, \ell)$ where
  - $T$ is the set of finite runs of $\mathcal{M}$ from $q$;
  - $\ell$ labels runs with the label of their last state.
Modelling reactive systems: finite-state transition systems

Formally:

**Definition**

- A **finite run** of $M$ from $q$ is a pair $(w, \ell)$ where
  - $w = (q_i)_{0 \leq i \leq n}$ is a finite sequence of states with
    - $q_0 = q$
    - $(q_i, q_{i+1}) \in R$ for all $0 \leq i \leq n - 1$;
  - $\ell$ labels states with atomic propositions;

- The **execution tree** of $M$ from $q$ is a pair $(T, \ell)$ where
  - $T$ is the set of finite runs of $M$ from $q$;
  - $\ell$ labels runs with the label of their last state.
Linear-time Temporal Logic (LTL)

atomic propositions: \( \phi, \psi, \ldots \)

boolean combinators: \( \neg \phi, \phi \lor \psi, \phi \land \psi, \ldots \)

temporal modalities: \( X \phi, \phi U \psi, \phi U \psi, \phi \text{ eventually} \)

\[ A Kripke \text{ structure satisfies } \phi \in \text{LTL} \text{ if all its infinite executions do:} \]
\[ M |\sigma = \phi \iff \forall \pi \in T M . \pi |\sigma = \phi. \]
Linear-time Temporal Logic (LTL)

- atomic propositions: \( \bigcirc, \bigcirc, \ldots \)
Linear-time Temporal Logic (LTL)

- atomic propositions: \( \bigcirc, \bigcirc, \ldots \)

- boolean combinators: \( \neg \varphi, \varphi \lor \psi, \varphi \land \psi, \ldots \)
Linear-time Temporal Logic (LTL)

- **atomic propositions:** \( \bigcirc, \bigcirc, \ldots \)

- **boolean combinators:** \( \neg \varphi, \varphi \lor \psi, \varphi \land \psi, \ldots \)

- **temporal modalities:**
  - \( X \varphi \) \hspace{1cm} \text{“next } \varphi\text{”}
  - \( \varphi \mathbf{U} \psi \) \hspace{1cm} \text{“} \varphi \text{ until } \psi\text{”}
Linear-time Temporal Logic (LTL)

- **atomic propositions:** \( \bigcirc, \odot, \ldots \)

- **boolean combinators:** \( \neg \varphi, \varphi \lor \psi, \varphi \land \psi, \ldots \)

- **temporal modalities:**
  
  \[
  \begin{align*}
  \mathbf{X} \varphi & \quad \xrightarrow{\varphi} \xrightarrow{\varphi} \xrightarrow{\varphi} \xrightarrow{\varphi} \xrightarrow{\varphi} \ldots \quad \text{“next } \varphi\text{”} \\
  \varphi \mathbf{U} \psi & \quad \xrightarrow{\varphi} \xrightarrow{\varphi} \xrightarrow{\psi} \xrightarrow{\varphi} \xrightarrow{\varphi} \ldots \quad \text{“} \varphi \text{ until } \psi\text{”} \\
  \text{true } \mathbf{U} \varphi & \equiv \mathbf{F} \varphi \quad \xrightarrow{\varphi} \xrightarrow{\varphi} \xrightarrow{\varphi} \xrightarrow{\varphi} \xrightarrow{\varphi} \ldots \quad \text{“eventually } \varphi\text{”} \\
  \neg \mathbf{F} \neg \varphi & \equiv \mathbf{G} \varphi \quad \xrightarrow{\varphi} \xrightarrow{\varphi} \xrightarrow{\varphi} \xrightarrow{\varphi} \xrightarrow{\varphi} \ldots \quad \text{“always } \varphi\text{”}
  \end{align*}
  \]
Linear-time Temporal Logic (LTL)

- **atomic propositions:** ○, ○, ...
- **boolean combinators:** ¬φ, φ ∨ ψ, φ ∧ ψ, ...
- **temporal modalities:**
  - Xφ
  - φ U ψ
  - true U φ ≡ F φ
  - ¬F ¬φ ≡ G φ

A Kripke structure satisfies φ ∈ LTL if all its infinite executions do:

\[ M \models \varphi \iff \forall \pi \in T_M. \pi \models \varphi. \]
Examples of LTL formulas

$$F \ floor_1$$

*the cabin eventually reaches the first floor*
Examples of LTL formulas

\[
\begin{align*}
F \text{ floor}_1 & \quad \text{the cabin eventually reaches the first floor} \\
G(\text{op}_1 \Rightarrow \text{serv}_1) & \quad \text{if the door is open, the cabin is present}
\end{align*}
\]
Examples of LTL formulas

\( \text{F} \, \text{floor}_1 \)  
*the cabin eventually reaches the first floor*

\( \text{G} \left( \text{op}_1 \Rightarrow \text{serv}_1 \right) \)  
*if the door is open, the cabin is present*

\( \text{G} \left( \text{req}_0 \Rightarrow \text{F} \, \text{serv}_0 \right) \)  
*any request is eventually served*
Examples of LTL formulas

\[ F \text{ floor}_1 \quad \text{the cabin eventually reaches the first floor} \]

\[ G(\text{op}_1 \Rightarrow \text{serv}_1) \quad \text{if the door is open, the cabin is present} \]

\[ G(\text{req}_0 \Rightarrow F \text{serv}_0) \quad \text{any request is eventually served} \]

\[ GF \text{ idle}_0 \quad \text{the button is off infinitely many times} \]
Examples of LTL formulas

- **F floor₁**  
  the cabin eventually reaches the first floor

- **G(op₁ ⇒ serv₁)**  
  if the door is open, the cabin is present

- **G(req₀ ⇒ F serv₀)**  
  any request is eventually served

- **G F idle₀**  
  the button is off infinitely many times

- **(G F serv₀) ⇒ (G F req₀)**  
  serve infinitely many times only if there are infinitely many requests
Satisfiability and model checking

Two main algorithmic problems

Satisfiability: Input: a formula $\phi$ in LTL; Output: yes if there exists a Kripke structure $M$ s.t. $M|\models \phi$; no otherwise.

Model checking: Input: a formula $\phi$ in LTL, and a Kripke structure $M$; Output: yes if $M|\models \phi$; no otherwise.
Satisfiability and model checking

Two main algorithmic problems

- **Satisfiability:**
  - Input: a formula $\phi$ in LTL;
  - Output:
    - *yes* if there exists a Kripke structure $M$ s.t. $M \models \phi$;
    - *no* otherwise.
Satisfiability and model checking

Two main algorithmic problems

- **Satisfiability:**
  - Input: a formula $\varphi$ in LTL;
  - Output:
    - yes if there exists a Kripke structure $M$ s.t. $M \models \varphi$;
    - no otherwise.

- **Model checking:**
  - Input: a formula $\varphi$ in LTL, and a Kripke structure $M$;
  - Output:
    - yes if $M \models \varphi$;
    - no otherwise.
Verifying LTL properties

Theorem (SC85)

LTL satisfiability and model checking are logspace-equivalent.

Proof (Sketch)

Encode behaviour of Kripke structure as an LTL formula:

\[ \text{exactly one state-proposition in } \{p, q, r\} \land (p \iff (p \lor q) \land (q \iff (p \lor q) \land ((p \lor q) \Rightarrow X (p \lor q)) \land (p \Rightarrow X (p \lor q))) \}

Lemma

\[ M, \neg \models \phi \iff \neg \phi \land \phi \text{ is satisfiable.} \]
Verifying LTL properties

<table>
<thead>
<tr>
<th>Theorem ([SC85])</th>
</tr>
</thead>
<tbody>
<tr>
<td>LTL satisfiability and model checking are logspace-equivalent.</td>
</tr>
</tbody>
</table>

Verifying LTL properties

**Theorem ([SC85])**

*LTL satisfiability and model checking are logspace-equivalent.*

**Proof (Sketch)**

Encode behaviour of Kripke structure as an LTL formula:

\[
\begin{align*}
G \left[ \text{exactly one state-proposition in } \{p, q, \varphi\} \right] \land (p \iff (\varphi \lor (\varphi \lor (q \iff (\varphi \lor (\varphi \rightarrow X (\varphi \land (\varphi \rightarrow X \varphi)))))))
\end{align*}
\]

**Lemma**

\[M, \varphi = \varphi \iff \neg \varphi \land \varphi \land \varphi\]

\[M\text{ is satisfiable.}\]

Verifying LTL properties

**Theorem ([SC85])**

LTL satisfiability and model checking are logspace-equivalent.

**Proof (Sketch)**

Encode behaviour of Kripke structure as an LTL formula:

\[
\text{G} \left[ \text{p, q} \right]
\]

Lemma

\[ M, \neg M = \varphi \iff \neg \varphi \wedge \Phi \]

\[ M \text{ is satisfiable.} \]

Verifying LTL properties

**Theorem ([SC85])**

*LTL satisfiability and model checking are logspace-equivalent.*

**Proof (Sketch)**

Encode behaviour of Kripke structure as an LTL formula:

\[
\begin{align*}
G & = \exists \text{ exactly one state-proposition in } \{p, q\} \\
\end{align*}
\]

\[
\begin{align*}
\enspace & \text{encoded as } \bigwedge \bigvee \bigwedge \bigvee \bigwedge \\
\end{align*}
\]

\[
\begin{align*}
\enspace & \text{encoded as } \bigwedge \bigvee \bigwedge \bigvee \bigwedge \\
\end{align*}
\]

Verifying LTL properties

Theorem ([SC85])

*LTL satisfiability and model checking are logspace-equivalent.*

Proof (Sketch)

Encode behaviour of Kripke structure as an LTL formula:

\[
G \left[ \left( p \Leftrightarrow (\bigcirc \lor \bigcirc) \right) \land \left( q \Leftrightarrow (\bigcirc \lor \bigcirc) \right) \right]
\]

where \( G \) represents the property that exactly one state-proposition in \( \{\bigcirc, \bigcirc, \bigcirc\} \) holds.

Verifying LTL properties

Theorem ([SC85])
LTL satisfiability and model checking are logspace-equivalent.

Proof (Sketch)
Encode behaviour of Kripke structure as an LTL formula:

\[
G \left[ \left( p \leftrightarrow (\bigcirc \lor \bigcirc) \right) \land \left( q \leftrightarrow (\bigcirc \lor \bigcirc) \right) \land \left( (\bigcirc \lor \bigcirc) \Rightarrow X(\bigcirc \lor \bigcirc) \right) \land \left( \bigcirc \Rightarrow X \bigcirc \right) \right]
\]

Verifying LTL properties

**Theorem ([SC85])**

*LTL satisfiability and model checking are logspace-equivalent.*

**Proof (Sketch)**

Encode behaviour of Kripke structure as an LTL formula:

\[
G \left[ (p \iff (\bigcirc \lor \bigcirc)) \land (q \iff (\bigcirc \lor \bigcirc)) \land \left[ (\bigcirc \lor \bigcirc) \Rightarrow \mathbf{X}(\bigcirc \lor \bigcirc) \right] \right.
\]

\[
\land \left[ \bigcirc \Rightarrow \mathbf{X} \bigcirc \right]
\]

**Lemma**

\[
\mathcal{M}, \bigcirc \models \varphi \iff \neg \varphi \land \bigcirc \land \Phi_{\mathcal{M}} \text{ is satisfiable.}
\]

Verifying LTL properties

**Theorem ([SC85])**

LTL satisfiability can be solved in polynomial space.

Verifying LTL properties

**Theorem ([SC85])**

*LTL satisfiability can be solved in polynomial space.*

**Proof (Sketch)**

**Lemma (Small-model property)**

*If $\varphi \in LTL$ is satisfiable, it holds in an exponential-size lasso-shaped model.*

Verifying LTL properties

Theorem ([SC85])

*LTL satisfiability can be solved in polynomial space.*

Proof (Sketch)

Lemma (Small-model property)

*If* $\varphi \in \text{LTL}$ *is satisfiable, it holds in an exponential-size lasso-shaped model.*

**formula:**

$$\varphi = \mathbf{G}(\mathbf{\circ} \Rightarrow \mathbf{F} \mathbf{\circ})$$

Verifying LTL properties

Theorem ([SC85])

**LTL satisfiability** can be solved in *polynomial space*.

Proof (Sketch)

Lemma (Small-model property)

*If* $\varphi \in \text{LTL}$ *is satisfiable, it holds in an exponential-size lasso-shaped model.*

**Formula:**

$$\varphi = \text{G}(*\Rightarrow* \text{F})*$$

**Subformulas:**

- $*, *$, $\text{F}*$, $\Rightarrow*$, $\text{G}(*\Rightarrow* \text{F})*$
Verifying LTL properties

Theorem ([SC85])

LTL satisfiability can be solved in polynomial space.

Proof (Sketch)

Lemma (Small-model property)

If $\varphi \in \text{LTL}$ is satisfiable, it holds in an exponential-size lasso-shaped model.

formula: $\varphi = G(\bigcirc \Rightarrow F \bigcirc)$

subformulas: $\bigcirc, \bigcirc, F \bigcirc, \bigcirc \Rightarrow F \bigcirc, G(\bigcirc \Rightarrow F \bigcirc)$

Verifying LTL properties

Theorem ([SC85])

LTL satisfiability can be solved in polynomial space.

Proof (Sketch)

Lemma (Small-model property)

If \( \varphi \in \text{LTL} \) is satisfiable, it holds in an exponential-size lasso-shaped model.

formula: \( \varphi = G(\circ \Rightarrow F \circ) \)

subformulas: \( \circ, \circ, F \circ, \circ \Rightarrow F \circ, G(\circ \Rightarrow F \circ) \)

same subformulas hold at both positions;
all \( U \)-formulas fulfilled inbetween

Verifying LTL properties

**Theorem ([SC85])**

LTL satisfiability can be solved in polynomial space.

**Proof (Sketch)**

**Lemma (Small-model property)**

If \( \varphi \in \text{LTL} \) is satisfiable, it holds in an exponential-size lasso-shaped model.

- **formula:** \( \varphi = G(\bigcirc \Rightarrow F \bigcirc) \)
- **subformulas:** \( \bigcirc, \bigcirc, F \bigcirc, \bigcirc \Rightarrow F \bigcirc, G(\bigcirc \Rightarrow F \bigcirc) \)

Verifying LTL properties

Theorem ([SC85])

LTL satisfiability can be solved in polynomial space.

Proof (Sketch)

Lemma (Small-model property)

If $\varphi \in LTL$ is satisfiable, it holds in an exponential-size lasso-shaped model.

Theorem ([SC85])

LTL model checking and satisfiability are PSPACE-complete.

Verifying LTL properties – automata-theoretic approach

Theorem ([VW86])

For any $\varphi \in \text{LTL}$, there exists an exponential-size Büchi automaton $A_\varphi$ such that

$$\forall w. \quad w \models \varphi \iff w \in \text{Lang}(A_\varphi).$$

Verifying LTL properties – automata-theoretic approach

**Theorem ([VW86])**

For any $\varphi \in LTL$, there exists an exponential-size Büchi automaton $A_{\varphi}$ such that

$$\forall w. \quad w \models \varphi \iff w \in \text{Lang}(A_{\varphi}).$$

**Example**

![Automaton Diagram]

Verifying LTL properties – automata-theoretic approach

Theorem ([VW86])

For any $\varphi \in \text{LTL}$, there exists an exponential-size Büchi automaton $A_\varphi$ such that

$$\forall w. \quad w \models \varphi \iff w \in \text{Lang}(A_\varphi).$$

Example

Verifying LTL properties – automata-theoretic approach

**Theorem ([VW86])**

For any $\varphi \in \text{LTL}$, there exists an exponential-size Büchi automaton $A_\varphi$ such that

$$\forall w. \quad w \models \varphi \iff w \in \text{Lang}(A_\varphi).$$

**Example**

![Diagram showing a Büchi automaton for a specific LTL formula. The automaton has states $q_0$ and $q_1$, with transitions labeled by propositions and the initial state marked.]
Verifying LTL properties – automata-theoretic approach

**Theorem ([VW86])**

For any $\varphi \in \text{LTL}$, there exists an exponential-size Büchi automaton $A_\varphi$ such that

$$\forall w. \ w \models \varphi \iff w \in \text{Lang}(A_\varphi).$$

**Example**

![Example diagram](image)

Verifying LTL properties – automata-theoretic approach

**Theorem ([VW86])**

For any $\varphi \in \text{LTL}$, there exists an exponential-size Büchi automaton $A_\varphi$ such that

$$\forall w. \quad w \models \varphi \iff w \in \text{Lang}(A_\varphi).$$

**Example**

\[\begin{array}{c}
\text{Example}\\
\begin{array}{c}
\text{q}_0 \\
\text{q}_1 \\
\end{array}
\end{array} \]

Verifying LTL properties – automata-theoretic approach

Theorem ([VW86])

For any \( \varphi \in LTL \), there exists an exponential-size Büchi automaton \( A_\varphi \) such that

\[
\forall w. \quad w \models \varphi \iff w \in \text{Lang}(A_\varphi).
\]

Example

\[G(\neg F)\]

Verifying LTL properties – automata-theoretic approach

**Theorem ([VW86])**

For any $\varphi \in \mathsf{LTL}$, there exists an exponential-size Büchi automaton $A_\varphi$ such that

$$\forall w. \ w \models \varphi \iff w \in \mathsf{Lang}(A_\varphi).$$

**Example**

![Automaton Diagram](image)

Verifying LTL properties – automata-theoretic approach

Theorem ([VW86])

For any $\varphi \in LTL$, there exists an exponential-size Büchi automaton $A_\varphi$ such that

$$\forall w. \ w \models \varphi \iff w \in \text{Lang}(A_\varphi).$$

Example

Verifying LTL properties – automata-theoretic approach

**Theorem ([VW86])**

For any $\varphi \in \text{LTL}$, there exists an exponential-size Büchi automaton $A_\varphi$ such that

$$\forall w. \quad w \models \varphi \iff w \in \text{Lang}(A_\varphi).$$

**Example**

![Büchi Automaton Example](image)

Verifying LTL properties – automata-theoretic approach

**Theorem ([VW86])**

For any $\varphi \in LTL$, there exists an exponential-size Büchi automaton $A_\varphi$ such that

$$\forall w. \ w \models \varphi \iff w \in \text{Lang}(A_\varphi).$$

**Example**

[Diagram of a Büchi automaton with states $q_0$ and $q_1$.]
Verifying LTL properties – automata-theoretic approach

Theorem ([VW86])

For any $\varphi \in LTL$, there exists an exponential-size Büchi automaton $A_\varphi$ such that

$$\forall w. \quad w \models \varphi \iff w \in \text{Lang}(A_\varphi).$$

Example

Verifying LTL properties – automata-theoretic approach

**Theorem ([VW86])**

For any $\varphi \in LTL$, there exists an exponential-size Büchi automaton $A_\varphi$ such that

$$\forall w. \quad w \models \varphi \iff w \in \text{Lang}(A_\varphi).$$

**Example**

Verifying LTL properties – automata-theoretic approach

**Theorem ([VW86])**

For any $\varphi \in LTL$, there exists an exponential-size Büchi automaton $A_\varphi$ such that

$$\forall w. \quad w \models \varphi \iff w \in \text{Lang}(A_\varphi).$$

**Example**

Theorem ([VW86])

For any $\varphi \in \text{LTL}$, there exists an exponential-size Büchi automaton $A_\varphi$ such that

$$\forall w. \quad w \models \varphi \iff w \in \text{Lang}(A_\varphi).$$

Example

Verifying LTL properties – automata-theoretic approach

Theorem ([VW86])

For any $\varphi \in LTL$, there exists an exponential-size Büchi automaton $A_\varphi$ such that

$$\forall w. \ w \models \varphi \iff w \in \text{Lang}(A_\varphi).$$

Example

\[
\begin{align*}
\neg \ 
\end{align*}
\]

Verifying LTL properties – automata-theoretic approach

**Theorem ([VW86])**

For any $\varphi \in LTL$, there exists an exponential-size *Büchi automaton* $A_\varphi$ such that

$$\forall w. \quad w \models \varphi \iff w \in \text{Lang}(A_\varphi).$$

**Proof (Sketch)**

- Use a slightly extended notion of subformula: $U; \lor (\land X (U))$.
- States of $A_\varphi$ = maximal consistent sets of subformulas.
- Transition $S \to S'$ whenever for all $X \psi \in S$, it holds $\psi \in S'$.
- Acceptance condition: infinitely often $\neg (U)$ (generalized Büchi condition).

Verifying LTL properties – automata-theoretic approach

Theorem ([VW86])

For any $\varphi \in LTL$, there exists an exponential-size Büchi automaton $A_\varphi$ such that

$$\forall w. \ w \models \varphi \iff w \in \text{Lang}(A_\varphi).$$

Proof (Sketch)

- use a slightly extended notion of subformula:

$$\bigcirc \ U \ \bigcirc \ \sim \ \bigcirc \ \lor \ (\bigcirc \ \land \ X(\bigcirc \ U \ \bigcirc)).$$
Verifying LTL properties – automata-theoretic approach

Theorem ([VW86])

For any $\varphi \in LTL$, there exists an exponential-size Büchi automaton $A_\varphi$ such that

$$\forall w. \ w \models \varphi \iff w \in \text{Lang}(A_\varphi).$$

Proof (Sketch)

- use a slightly extended notion of subformula:

  $$\bigcirc \ U \bigcirc \sim \bigcirc \lor \ (\bigcirc \land X(\bigcirc \ U \bigcirc)).$$

- states of $A_\varphi$ = maximal consistent sets of subformulas;
- transition $S \rightarrow S'$ whenever for all $X \psi \in S$, it holds $\psi \in S'$;

Verifying LTL properties – automata-theoretic approach

Theorem ([VW86])

For any $\varphi \in \text{LTL}$, there exists an exponential-size Büchi automaton $A_{\varphi}$ such that

$$\forall w. \quad w \models \varphi \iff w \in \text{Lang}(A_{\varphi}).$$

Proof (Sketch)

- use a slightly extended notion of subformula:
  $$\bigcirc \mathbf{U} \bigcirc \leadsto \bigcirc \lor (\bigcirc \land \mathbf{X}(\bigcirc \mathbf{U} \bigcirc)).$$

- states of $A_{\varphi}$ = maximal consistent sets of subformulas;
- transition $S \rightarrow S'$ whenever for all $\mathbf{X} \psi \in S$, it holds $\psi \in S'$;
- acceptance condition: infinitely often $\neg (\bigcirc \mathbf{U} \bigcirc)$
  (generalized Büchi condition)

Verifying LTL properties – automata-theoretic approach

Theorem ([VW86])

For any $\varphi \in LTL$, there exists an exponential-size Büchi automaton $A_\varphi$ such that

$$\forall w. \quad w \models \varphi \iff w \in \text{Lang}(A_\varphi).$$

Theorem ([VW86])

Model checking LTL can be performed in time $2^{O(|\varphi|)} \cdot O(|M|)$.  

Verifying LTL properties – using alternating automata

Theorem ([MSS88,Var94])

For any $\varphi \in LTL$, there exists a linear-size weak alternating Büchi automaton $A_\varphi$ such that

$$\forall w. \ w \models \varphi \iff w \in \text{Lang}(A_\varphi).$$

[MSS88] Muller, Saoudi, Schupp. Weak alternating automata give a simple explanation ... LICS, 1988.
Verifying LTL properties – using alternating automata

Theorem ([MSS88,Var94])

For any $\varphi \in \text{LTL}$, there exists a linear-size weak alternating Büchi automaton $A_\varphi$ such that

$$\forall w. \ w \models \varphi \iff w \in \text{Lang}(A_\varphi).$$

Example

![Automaton Diagram]

[MSS88] Muller, Saoudi, Schupp. Weak alternating automata give a simple explanation ... LICS, 1988.
Verifying LTL properties – using alternating automata

Theorem ([MSS88,Var94])

For any $\varphi \in LTL$, there exists a linear-size weak alternating Büchi automaton $A_\varphi$ such that

$$\forall w. \ w \models \varphi \iff w \in \text{Lang}(A_\varphi).$$

Example

[Image of a transition diagram]

[MSS88] Muller, Saoudi, Schupp. Weak alternating automata give a simple explanation ... LICS, 1988.
Verifying LTL properties – using alternating automata

Theorem ([MSS88, Var94])

For any $\varphi \in \text{LTL}$, there exists a linear-size weak alternating Büchi automaton $A_\varphi$ such that

$$\forall w. \quad w \models \varphi \iff w \in \text{Lang}(A_\varphi).$$

Example

[MS88] Muller, Saoudi, Schupp. Weak alternating automata give a simple explanation ... LICS, 1988.
Verifying LTL properties – using alternating automata

Theorem ([MSS88,Var94])

For any $\varphi \in LTL$, there exists a linear-size weak alternating Büchi automaton $A_\varphi$ such that

$$\forall w. \quad w \models \varphi \iff w \in \text{Lang}(A_\varphi).$$

Example

![Diagram of an alternating automaton]

[MSS88] Muller, Saoudi, Schupp. Weak alternating automata give a simple explanation... LICS, 1988.
Verifying LTL properties – using alternating automata

**Theorem ([MSS88,Var94])**

For any $\varphi \in \text{LTL}$, there exists a linear-size **weak alternating Büchi automaton** $A_\varphi$ such that

$$\forall w. \quad w \models \varphi \iff w \in \text{Lang}(A_\varphi).$$

**Example**

![Example Diagram]

[MSS88] Muller, Saoudi, Schupp. Weak alternating automata give a simple explanation ... LICS, 1988.
Verifying LTL properties – using alternating automata

**Theorem ([MSS88,Var94])**

For any $\varphi \in \text{LTL}$, there exists a linear-size weak alternating Büchi automaton $A_\varphi$ such that

$$\forall w. \quad w \models \varphi \iff w \in \text{Lang}(A_\varphi).$$

**Example**

![Diagram of a weak alternating Büchi automaton]

[MSS88] Muller, Saoudi, Schupp. Weak alternating automata give a simple explanation ... LICS, 1988.

Verifying LTL properties – using alternating automata

Theorem ([MSS88, Var94])

For any $\varphi \in \text{LTL}$, there exists a linear-size weak alternating Büchi automaton $A_\varphi$ such that

$$\forall w. \ w \models \varphi \iff w \in \text{Lang}(A_\varphi).$$

Example

[[MSS88] Muller, Saoudi, Schupp. Weak alternating automata give a simple explanation ... LICS, 1988.
[Var94] Vardi. Nontraditional applications of automata theory. TACS, 1994.]
Verifying LTL properties – using alternating automata

Theorem ([MSS88,Var94])

For any $\varphi \in LTL$, there exists a linear-size weak alternating Büchi automaton $A_\varphi$ such that

$$\forall w. \ w \models \varphi \iff w \in \text{Lang}(A_\varphi).$$

Example

[Example Diagram]

[MSS88] Muller, Saoudi, Schupp. Weak alternating automata give a simple explanation ... LICS, 1988.
Verifying LTL properties – using alternating automata

**Theorem ([MSS88,Var94])**

For any $\varphi \in \text{LTL}$, there exists a linear-size weak alternating Büchi automaton $A_\varphi$ such that

$$\forall w. \quad w \models \varphi \iff w \in \text{Lang}(A_\varphi).$$

**Example**

[MSS88] Muller, Saoudi, Schupp. Weak alternating automata give a simple explanation ... LICS, 1988.

Verifying LTL properties – using alternating automata

Theorem ([MSS88, Var94])

For any $\varphi \in LTL$, there exists a linear-size weak alternating Büchi automaton $A_\varphi$ such that

$$\forall w. \ w \models \varphi \iff w \in \text{Lang}(A_\varphi).$$

Example

[MSS88] Muller, Saoudi, Schupp. Weak alternating automata give a simple explanation ... LICS, 1988.
Verifying LTL properties – using alternating automata

Theorem ([MSS88,Var94])

For any $\varphi \in \text{LTL}$, there exists a linear-size weak alternating Büchi automaton $A_\varphi$ such that

$$\forall w. \; w \models \varphi \iff w \in \text{Lang}(A_\varphi).$$

Example

[MSS88] Muller, Saoudi, Schupp. Weak alternating automata give a simple explanation ... LICS, 1988.
Verifying LTL properties – using alternating automata

Theorem ([MSS88,Var94])

For any $\varphi \in \text{LTL}$, there exists a linear-size weak alternating Büchi automaton $A_\varphi$ such that

$$\forall w. \ w \models \varphi \iff w \in \text{Lang}(A_\varphi).$$

Example

[MS88] Muller, Saoudi, Schupp. Weak alternating automata give a simple explanation ... LICS, 1988.
Verifying LTL properties – using alternating automata

Theorem ([MSS88,Var94])

For any $\varphi \in \text{LTL}$, there exists a linear-size weak alternating Büchi automaton $A_\varphi$ such that

$$\forall w. \ w \models \varphi \iff w \in \text{Lang}(A_\varphi).$$

Example

[MSS88] Muller, Saoudi, Schupp. Weak alternating automata give a simple explanation ... LICS, 1988.
Verifying LTL properties – using alternating automata

Theorem ([MSS88,Var94])

For any $\varphi \in \text{LTL}$, there exists a linear-size weak alternating Büchi automaton $A_\varphi$ such that

$$\forall w. \ w \models \varphi \iff w \in \text{Lang}(A_\varphi).$$

Example

[MSS88] Muller, Saoudi, Schupp. Weak alternating automata give a simple explanation ... LICS, 1988.

Verifying LTL properties – using alternating automata

Theorem ([MSS88, Var94])

For any \( \varphi \in \text{LTL} \), there exists a linear-size weak alternating Büchi automaton \( A_\varphi \) such that

\[
\forall w. \quad w \models \varphi \iff w \in \text{Lang}(A_\varphi).
\]

Example

[MSS88] Muller, Saoudi, Schupp. Weak alternating automata give a simple explanation ... LICS, 1988.
Verifying LTL properties – using alternating automata

**Theorem ([MSS88, Var94])**

For any $\varphi \in LTL$, there exists a linear-size weak alternating Büchi automaton $A_\varphi$ such that

$$\forall w. \ w \models \varphi \iff w \in \text{Lang}(A_\varphi).$$

**Example**

[MSS88] Muller, Saoudi, Schupp. Weak alternating automata give a simple explanation ... LICS, 1988.
Verifying LTL properties – using alternating automata

Theorem ([MSS88,Var94])

For any $\varphi \in \text{LTL}$, there exists a linear-size weak alternating Büchi automaton $A_\varphi$ such that

$$\forall w. \quad w \models \varphi \iff w \in \text{Lang}(A_\varphi).$$

Example

[MSS88] Muller, Saoudi, Schupp. Weak alternating automata give a simple explanation ... LICS, 1988.
Verifying LTL properties – using alternating automata

Theorem ([MSS88,Var94])

For any $\varphi \in LTL$, there exists a linear-size weak alternating Büchi automaton $A_\varphi$ such that

$$\forall w. \quad w \models \varphi \iff w \in \text{Lang}(A_\varphi).$$

Proof (Sketch)

- one state per temporal modality, atomic proposition, and negations thereof;

[MSS88] Muller, Saoudi, Schupp. Weak alternating automata give a simple explanation ... LICS, 1988.
Verifying LTL properties – using alternating automata

**Theorem ([MSS88,Var94])**

For any $\varphi \in LTL$, there exists a linear-size weak alternating Büchi automaton $A_{\varphi}$ such that

$$\forall w. \ w \models \varphi \iff w \in \text{Lang}(A_{\varphi}).$$

**Proof (Sketch)**

- one state per temporal modality, atomic proposition, and negations thereof;
- transitions defined as boolean formulas:

$$\delta(\psi_1 \mathbf{U} \psi_2, \bigcirc) = \delta(\psi_2, \bigcirc) \lor [\delta(\psi_1, \bigcirc) \land \psi_1 \mathbf{U} \psi_2].$$

[MSS88] Muller, Saoudi, Schupp. Weak alternating automata give a simple explanation ... LICS, 1988.
Theorem ([MSS88,Var94])

For any $\varphi \in LTL$, there exists a linear-size weak alternating Büchi automaton $A_\varphi$ such that

$$\forall w. \quad w \models \varphi \iff w \in \text{Lang}(A_\varphi).$$

Proof (Sketch)

- one state per temporal modality, atomic proposition, and negations thereof;
- transitions defined as boolean formulas:
  $$\delta(\psi_1 \text{ U } \psi_2, \bigcirc) = \delta(\psi_2, \bigcirc) \lor [\delta(\psi_1, \bigcirc) \land \psi_1 \text{ U } \psi_2].$$
- acceptance condition: Büchi condition over non-until states.

[MSS88] Muller, Saoudi, Schupp. Weak alternating automata give a simple explanation ... LICS, 1988.
When are two Kripke structures indistinguishable by LTL?
Behavioural equivalence for LTL

When are two Kripke structures indistinguishable by LTL?

Example
Behavioral equivalence for LTL

**Theorem**

Two *finitely branching* Kripke structures satisfy the same LTL formulas if, and only if, they generate the same sets of traces.

Proof (Sketch)

**Lemma**

If any finite trace of $M$ can be generated by $N$, then $\text{InfTraces}(M) \subseteq \text{InfTraces}(N)$ (for finitely branching Kripke structures $M$ and $N$).
Theorem

Two finitely branching Kripke structures satisfy the same LTL formulas if, and only if, they generate the same sets of traces.

Proof (Sketch)

Lemma

If any finite trace of $\mathcal{M}$ can be generated by $\mathcal{N}$, then

$$\text{InfTraces}(\mathcal{M}) \subseteq \text{InfTraces}(\mathcal{N})$$

(for finitely branching Kripke structures $\mathcal{M}$ and $\mathcal{N}$).
Theorem

Two \textit{finitely branching} Kripke structures satisfy the same LTL formulas if, and only if, they generate the same sets of traces.
Theorem

Two finitely branching Kripke structures satisfy the same LTL formulas if, and only if, they generate the same sets of traces.

Counter-example

$$w \in \text{Traces}(M) \iff \exists i > 0. \ w(2i) = \circ$$
Computation-Tree Logic (CTL*)

atomic propositions: \( \phi \), \( \psi \), ...

boolean combinators: \( \neg \phi \), \( \phi \lor \psi \), \( \phi \land \psi \), ...

temporal modalities: \( \mathcal{X} \phi \) \( \mathcal{E} \phi \) \( \mathcal{A} \phi \)

path quantifiers: \( \mathcal{E} \phi \), \( \mathcal{A} \phi \)
Computation-Tree Logic (CTL*)

- atomic propositions: $\bullet, \bullet, \ldots$

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

- temporal modalities:
  - $X \varphi$
  - $\varphi U \psi$

```
X \varphi               \varphi \rightarrow \bullet \rightarrow \bullet \rightarrow \bullet \rightarrow \bullet \rightarrow \bullet \rightarrow \bullet

$\varphi$ until $\psi$
```

```
\varphi \rightarrow \varphi \rightarrow \psi \rightarrow \bullet \rightarrow \bullet \rightarrow \bullet
```

“next $\varphi$”

“$\varphi$ until $\psi$”
Computation-Tree Logic (CTL*)

- atomic propositions: $\odot$, $\ominus$, ...

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

- temporal modalities:
  - $X \varphi$
  - $\varphi \mathbf{U} \psi$

- path quantifiers:
Examples of CTL formulas

In CTL, each temporal modality must be in the immediate scope of a path quantifier.
Examples of CTL formulas

In CTL, each temporal modality must be in the immediate scope of a path quantifier.

$$\forall F \text{floor}_1$$ the cabin eventually reaches the first floor
In CTL, each temporal modality must be in the immediate scope of a path quantifier.

\[ \text{A} \ F \ \text{floor}_1 \quad \text{the cabin eventually reaches the first floor} \]

\[ \text{A} \ G (\text{op}_1 \Rightarrow \text{serv}_1) \quad \text{if the door is open, the cabin is present} \]
Examples of CTL formulas

In CTL, each temporal modality must be in the immediate scope of a path quantifier.

\[
\text{A } F 	ext{ floor}_1 \quad \text{the cabin eventually reaches the first floor}
\]

\[
\text{A } G (\text{op}_1 \implies \text{serv}_1) \quad \text{if the door is open, the cabin is present}
\]

\[
\text{A } G (\text{req}_0 \implies \text{A } F \text{ serv}_0) \quad \text{any request is eventually served}
\]
Examples of CTL formulas

In CTL, each temporal modality must be in the immediate scope of a path quantifier.

\[ AF(\text{floor}_1) \quad \text{the cabin eventually reaches the first floor} \]

\[ AG(\text{op}_1 \Rightarrow \text{serv}_1) \quad \text{if the door is open, the cabin is present} \]

\[ AG(\text{req}_0 \Rightarrow AF\text{serv}_0) \quad \text{any request is eventually served} \]

\[ AG(\text{req}_0 \Rightarrow EF\text{serv}_0) \quad \text{any request can eventually be served} \]
Examples of CTL formulas

In CTL, each temporal modality must be in the immediate scope of a path quantifier.

AF(floor₁)  
the cabin eventually reaches the first floor

AG(op₁ ⇒ serv₁)  
if the door is open, the cabin is present

AG(req₀ ⇒ AF(serv₀))  
any request is eventually served

AG(req₀ ⇒ EF(serv₀))  
any request can eventually be served

AG(AF(idle₀))  
the button is off infinitely often
Expressiveness of CTL

Theorem ([EH86])

CTL cannot express the following property:

*there is a path visiting infinitely many times.*

Expressiveness of CTL

**Theorem ([EH86])**

CTL cannot express the following property:

*there is a path visiting infinitely many times.*

**Example**

\[ AGAF \] visited infinitely many times along all branches

Expressiveness of CTL

Theorem ([EH86])

CTL cannot express the following property:

there is a path visiting \( \circ \) infinitely many times.

Example

AGAF \( \circ \) visited infinitely many times along all branches

Expressiveness of CTL

Theorem ([EH86])

CTL cannot express the following property:

there is a path visiting infinitely many times.

Example

AGAF visited infinitely many times along all branches

EGEF
Expressiveness of CTL

Theorem ([EH86])

CTL cannot express the following property:

*there is a path visiting infinitely many times.*

Example

\[ \text{AGAF} \quad \text{visited infinitely many times along all branches} \]

\[ \text{EGEF} \quad \text{there is a path along which can always be visited} \]
Expressiveness of CTL

**Theorem ([EH86])**

CTL cannot express the following property:

*there is a path visiting infinitely many times.*

**Example**

- **AGAF** visited infinitely many times along all branches
- **EGEF** there is a path along which can always be visited
- **EGAF**
Expressiveness of CTL

Theorem ([EH86])

CTL cannot express the following property:

\[ \text{there is a path visiting \(\infty\) infinitely many times.} \]

Example

\[ \text{AGAF} \quad \text{visited infinitely many times along all branches} \]

\[ \text{EGEF} \quad \text{there is a path along which can always be visited} \]

\[ \text{EGAF} \quad \text{there is a path along which cannot be avoided} \]

Expressiveness of CTL

Theorem ([EH86])

CTL cannot express the following property:

there is a path visiting \(\bigcirc\) infinitely many times.

Proof

\[ \alpha_i \not\models \text{EGF} \]

\[ \alpha'_i \models \text{EGF} \]
Expressiveness of CTL

**Theorem ([EH86])**

*CTL cannot express the following property:*

*there is a path visiting $\bigcirc$ infinitely many times.*

**Proof**

![Diagram of CTL formulas](image)

**Lemma**

$\alpha_i$ and $\alpha'_i$ satisfy the same CTL formulas of length at most $i$.

Expressiveness of CTL

Theorem

Modalities $\textbf{EX}$, $\textbf{EU}$ and $\textbf{EG}$ are sufficient to express any CTL formula.
Expressiveness of CTL

Theorem
Modalities $\mathbf{EX}$, $\mathbf{E} \mathbf{U}$ and $\mathbf{EG}$ are sufficient to express any CTL formula.

Proof

$\mathbf{AX} \equiv \neg \mathbf{EX} \neg \mathbf{A} \mathbf{U} \equiv \neg \mathbf{E} (\neg \mathbf{A}) \mathbf{U} (\neg \mathbf{A} \land \neg \mathbf{A}) \land \neg \mathbf{EG} \neg \mathbf{A}$

How to falsify $\mathbf{U}$: never; otherwise: pick earliest; there must be a $\neg \mathbf{A}$ before.
Theorem

Modalities $\textbf{EX}$, $\textbf{E U}$ and $\textbf{EG}$ are sufficient to express any CTL formula.

Proof

$$\textbf{AX} \equiv \neg \textbf{EX} \neg \neg \neg$$
Expressiveness of CTL

**Theorem**

Modalities $\text{EX}$, $\text{E U}$ and $\text{EG}$ are sufficient to express any CTL formula.

**Proof**

$$\text{AX} \triangleq \neg \text{EX} \neg \star$$

$$\text{A} \star \text{U} \bullet \equiv$$
Expressiveness of CTL

Theorem

Modalities $\text{EX}$, $\text{E U}$ and $\text{EG}$ are sufficient to express any CTL formula.

Proof

$\text{AX} \equiv \neg \text{EX} \neg \bigcirc$

$\text{A} \bigcirc \text{U} \bigcirc \equiv$

How to falsify $\bigcirc \text{U} \bigcirc$:
Theorem

Modalities $\neg EX$, $E U$ and $EG$ are sufficient to express any CTL formula.

Proof

$$AX \equiv \neg EX \neg$$
$$A U \equiv$$

How to falsify $U$:
- never $U$;
Expressiveness of CTL

**Theorem**

Modalities $\mathbf{EX}$, $\mathbf{E} \mathbf{U}$ and $\mathbf{EG}$ are sufficient to express any CTL formula.

**Proof**

$$AX \equiv \neg EX \neg$$

$$A\square U \equiv$$

How to falsify $\square U$:

- never $\square$;
- otherwise:
  - pick earliest $\square$;
Expressiveness of CTL

**Theorem**

Modalities $\text{EX}$, $\text{EU}$ and $\text{EG}$ are sufficient to express any CTL formula.

**Proof**

\[
\begin{align*}
\text{AX} & \equiv \neg \text{EX} \neg \bigcirc \\
\text{A} \bigcirc \text{U} \bigcirc & \equiv
\end{align*}
\]

How to falsify $\bigcirc \text{U} \bigcirc$:

1. never $\bigcirc$;
2. otherwise:
   1. pick earliest $\bigcirc$;
   2. there must be a $\neg \bigcirc$ before.
Expressiveness of CTL

**Theorem**

Modalities $\text{EX}$, $\text{E U}$ and $\text{EG}$ are sufficient to express any CTL formula.

**Proof**

$\text{AX} \equiv \neg \text{EX} \neg \Box$

$\text{A} \Box \text{U} \Box \equiv \neg \text{E} (\neg \Box) \text{ U} (\neg \Box \land \neg \Box) \land \neg \text{EG} \neg \Box$

How to falsify $\Box \text{U} \Box$:

- never $\Box$;
- otherwise:
  - pick earliest $\Box$;
  - there must be a $\neg \Box$ before.
Verifying CTL properties

Theorem ([CE81, QS82])

CTL model checking is PTIME-complete.

Verifying CTL properties

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

*CTL model checking is PTIME-complete.*

**Proof**

label states with the subformulas they satisfy.

---


Verifying CTL properties

Theorem ([CE81, QS82])

CTL model checking is PTIME-complete.

Proof

label states with the subformulas they satisfy.

E F

is reachable

Verifying CTL properties

Theorem ([CE81, QS82])

CTL model checking is PTIME-complete.

Proof

label states with the subformulas they satisfy.

E F  is reachable

Verifying CTL properties

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

\textit{CTL model checking is PTIME-complete.}

**Proof**

Label states with the subformulas they satisfy.

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


Verifying CTL properties

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

*CTL model checking is PTIME-complete.*

**Proof**

label states with the subformulas they satisfy.

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


Verifying CTL properties

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

*CTL model checking is PTIME-complete.*

**Proof**

Label states with the subformulas they satisfy.

\[ \text{EG}(\neg \, \bigcirc \land \, \text{EF} \, \bigcirc) \]

There is a path along which \( \bigcirc \) is always reachable, but never reached.

Verifying CTL properties

Theorem ([CE81,QS82])

\textit{CTL model checking is PTIME-complete.}

Proof

label states with the subformulas they satisfy.

\[ \text{EG}(\neg p \land \text{EF} p) \] there is a path along which $p$ is always reachable, but never reached

Verifying CTL properties

Theorem ([CE81, QS82])

**CTL model checking is PTIME-complete.**

Proof

Label states with the subformulas they satisfy.

\[
EG(\neg \diamond p \land EF p)
\]

there is a path along which \( p \) is always reachable, but never reached


Verifying CTL properties

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

*CTL model checking is PTIME-complete.*

**Proof**

Label states with the subformulas they satisfy.

\[ \text{EG}(\neg \bigcirc \land \text{EF} \bigcirc) \]

there is a path along which \( \bigcirc \) is always reachable, but never reached

Verifying CTL properties

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

*CTL model checking is PTIME-complete.*

**Theorem ([KVW94])**

*CTL model checking on product structures is PSPACE-complete.*


Verifying CTL properties – automata-theoretic approach

Tree automata

Verifying CTL properties – automata-theoretic approach

Tree automata

Verifying CTL properties – automata-theoretic approach

Tree automata

\[
\begin{align*}
\delta(q_0, \bullet) &= (q_0, q_1) \lor (q_1, q_0) \\
\delta(q_0, \circ) &= (q_1, q_1) \\
\delta(q_0, \circledast) &= (q_2, q_2) \\
\delta(q_1, \circledast) &= (q_1, q_1) \\
\delta(q_2, \circledast) &= (q_2, q_2)
\end{align*}
\]

Verifying CTL properties – automata-theoretic approach

Tree automata

\[ \delta(q_0, \circ) = (q_0, q_1) \lor (q_1, q_0) \]
\[ \delta(q_0, \bullet) = (q_1, q_1) \]
\[ \delta(q_0, \text{gray}) = (q_2, q_2) \]
\[ \delta(q_1, \star) = (q_1, q_1) \]
\[ \delta(q_2, \star) = (q_2, q_2) \]

Verifying CTL properties – automata-theoretic approach

Tree automata

\[ \delta(q_0, \bullet) = (q_0, q_1) \lor (q_1, q_0) \]
\[ \delta(q_0, \circ) = (q_1, q_1) \]
\[ \delta(q_0, \square) = (q_2, q_2) \]
\[ \delta(q_1, \star) = (q_1, q_1) \]
\[ \delta(q_2, \star) = (q_2, q_2) \]

Verifying CTL properties – automata-theoretic approach

Tree automata

\[ \delta(q_0, \text{blue}) = (q_0, q_1) \lor (q_1, q_0) \]
\[ \delta(q_0, \text{orange}) = (q_1, q_1) \]
\[ \delta(q_0, \text{gray}) = (q_2, q_2) \]
\[ \delta(q_1, \text{black}) = (q_1, q_1) \]
\[ \delta(q_2, \text{black}) = (q_2, q_2) \]

Verifying CTL properties – automata-theoretic approach

Tree automata

\[ \delta(q_0, \bullet) = (q_0, q_1) \lor (q_1, q_0) \]
\[ \delta(q_0, \circ) = (q_1, q_1) \]
\[ \delta(q_0, \star) = (q_2, q_2) \]
\[ \delta(q_1, \square) = (q_1, q_1) \]
\[ \delta(q_2, \odot) = (q_2, q_2) \]

Verifying CTL properties – automata-theoretic approach

Tree automata

\[
\begin{align*}
\delta(q_0, \bullet) &= (q_0, q_1) \lor (q_1, q_0) \\
\delta(q_0, \circ) &= (q_1, q_1) \\
\delta(q_0, \star) &= (q_2, q_2) \\
\delta(q_1, \star) &= (q_1, q_1) \\
\delta(q_2, \star) &= (q_2, q_2)
\end{align*}
\]

Verifying CTL properties – automata-theoretic approach

This automaton corresponds to\[\delta(q_0, \circ) = (q_0, q_1) \lor (q_1, q_0)\]
\[\delta(q_0, \bullet) = (q_1, q_1)\]
\[\delta(q_0, \square) = (q_2, q_2)\]
\[\delta(q_1, \star) = (q_1, q_1)\]
\[\delta(q_2, \triangle) = (q_2, q_2)\]

Verifying CTL properties – automata-theoretic approach

Tree automata

\[
\begin{align*}
\delta(q_0, \circ) &= (q_0, q_1) \lor (q_1, q_0) \\
\delta(q_0, \bullet) &= (q_1, q_1) \\
\delta(q_0, \ast) &= (q_2, q_2) \\
\delta(q_1, \star) &= (q_1, q_1) \\
\delta(q_2, \oplus) &= (q_2, q_2)
\end{align*}
\]

Verifying CTL properties – automata-theoretic approach

Tree automata

\[ \delta(q_0, \bigcirc) = (q_0, q_1) \lor (q_1, q_0) \]
\[ \delta(q_0, \bullet) = (q_1, q_1) \]
\[ \delta(q_0, \star) = (q_2, q_2) \]
\[ \delta(q_1, \bigcirc) = (q_1, q_1) \]
\[ \delta(q_2, \star) = (q_2, q_2) \]

This automaton corresponds to \( E \bigcirc U \)
Verifying CTL properties – automata-theoretic approach

Alternating tree automata

Verifying CTL properties – automata-theoretic approach

Alternating tree automata

\[
\begin{align*}
\delta(q_0, \neg) &= \bot \\
\delta(q_1, \star) &= (q_1, q_2) \\
\delta(q_2, \star) &= (q_2, q_2) \\
\delta(q_3, \neg) &= (q_3, q_3)
\end{align*}
\]

Detailled explanations during François' lectures.

Verifying CTL properties – automata-theoretic approach

Alternating tree automata

\[
\delta(q_0, \bigcirc) = [(q_1, q_2) \lor (q_2, q_1)] \land (q_3, q_3)
\]

\[
\delta(q_0, \neg \bigcirc) = \bot
\]

\[
\delta(q_1, \bigcirc) = (q_1, q_2) \lor (q_2, q_1)
\]

\[
\delta(q_1, \neg \bigcirc) = \bot
\]

\[
\delta(q_2, \star) = (q_2, q_2)
\]

\[
\delta(q_3, \bigcirc) = (q_2, q_2)
\]

\[
\delta(q_3, \neg \bigcirc) = (q_3, q_3)
\]

Verifying CTL properties – automata-theoretic approach

Alternating tree automata

\[
\delta(q_0, \bigcirc) = [(q_1, q_2) \lor (q_2, q_1)] \land (q_3, q_3) \\
\delta(q_0, \neg \bigcirc) = \bot \\
\delta(q_1, \bigcirc) = (q_1, q_2) \lor (q_2, q_1) \\
\delta(q_1, \neg \bigcirc) = \bot \\
\delta(q_2, \bullet) = (q_2, q_2) \\
\delta(q_3, \bigcirc) = (q_2, q_2) \\
\delta(q_3, \neg \bigcirc) = (q_3, q_3)
\]

Verifying CTL properties – automata-theoretic approach

Alternating tree automata

\[ \delta(q_0, \bigcirc) = [(q_1, q_2) \lor (q_2, q_1)] \land (q_3, q_3) \]
\[ \delta(q_0, \neg \bigcirc) = \bot \]
\[ \delta(q_1, \bigcirc) = (q_1, q_2) \lor (q_2, q_1) \]
\[ \delta(q_1, \neg \bigcirc) = \bot \]
\[ \delta(q_2, \bigstar) = (q_2, q_2) \]
\[ \delta(q_3, \bigcirc) = (q_2, q_2) \]
\[ \delta(q_3, \neg \bigcirc) = (q_3, q_3) \]

Verifying CTL properties – automata-theoretic approach

Alternating tree automata

\[ \delta(q_0, \bullet) = [(q_1, q_2) \lor (q_2, q_1)] \land (q_3, q_3) \]
\[ \delta(q_0, \neg \bullet) = \bot \]
\[ \delta(q_1, \bullet) = (q_1, q_2) \lor (q_2, q_1) \]
\[ \delta(q_1, \neg \bullet) = \bot \]
\[ \delta(q_2, \star) = (q_2, q_2) \]
\[ \delta(q_3, \bullet) = (q_2, q_2) \]
\[ \delta(q_3, \neg \bullet) = (q_3, q_3) \]

Verifying CTL properties – automata-theoretic approach

Alternating tree automata

\[ \delta(q_0, \bigcirc) = [(q_1, q_2) \lor (q_2, q_1)] \land (q_3, q_3) \]
\[ \delta(q_0, \neg \bigcirc) = \bot \]
\[ \delta(q_1, \bigcirc) = (q_1, q_2) \lor (q_2, q_1) \]
\[ \delta(q_1, \neg \bigcirc) = \bot \]
\[ \delta(q_2, \star) = (q_2, q_2) \]
\[ \delta(q_3, \bigcirc) = (q_2, q_2) \]
\[ \delta(q_3, \neg \bigcirc) = (q_3, q_3) \]

Verifying CTL properties – automata-theoretic approach

Alternating tree automata

\[
\delta(q_0, \bigcirc) = [(q_1, q_2) \lor (q_2, q_1)] \land (q_3, q_3)
\]
\[
\delta(q_0, \neg \bigcirc) = \bot
\]
\[
\delta(q_1, \bigcirc) = (q_1, q_2) \lor (q_2, q_1)
\]
\[
\delta(q_1, \neg \bigcirc) = \bot
\]
\[
\delta(q_2, \bigstar) = (q_2, q_2)
\]
\[
\delta(q_3, \bigcirc) = (q_2, q_2)
\]
\[
\delta(q_3, \neg \bigcirc) = (q_3, q_3)
\]

Verifying CTL properties – automata-theoretic approach

Alternating tree automata

\[
\delta(q_0, \bigcirc) = [(q_1, q_2) \lor (q_2, q_1)] \land (q_3, q_3)
\]

\[
\delta(q_0, \neg \bigcirc) = \bot
\]

\[
\delta(q_1, \bigcirc) = (q_1, q_2) \lor (q_2, q_1)
\]

\[
\delta(q_1, \neg \bigcirc) = \bot
\]

\[
\delta(q_2, \star) = (q_2, q_2)
\]

\[
\delta(q_3, \bigcirc) = (q_2, q_2)
\]

\[
\delta(q_3, \neg \bigcirc) = (q_3, q_3)
\]

Verifying CTL properties – automata-theoretic approach

Alternating tree automata

\[ \delta(q_0, \bigcirc) = [(q_1, q_2) \lor (q_2, q_1)] \land (q_3, q_3) \]

\[ \delta(q_0, \lnot \bigcirc) = \bot \]

\[ \delta(q_1, \bigcirc) = (q_1, q_2) \lor (q_2, q_1) \]

\[ \delta(q_1, \lnot \bigcirc) = \bot \]

\[ \delta(q_2, \bigstar) = (q_2, q_2) \]

\[ \delta(q_3, \bigcirc) = (q_2, q_2) \]

\[ \delta(q_3, \lnot \bigcirc) = (q_3, q_3) \]

Verifying CTL properties – automata-theoretic approach

Alternating tree automata

\[
\begin{align*}
\delta(q_0, \bigcirc) &= \left[ (q_1, q_2) \lor (q_2, q_1) \right] \land (q_3, q_3) \\
\delta(q_0, \neg \bigcirc) &= \perp \\
\delta(q_1, \bigcirc) &= (q_1, q_2) \lor (q_2, q_1) \\
\delta(q_1, \neg \bigcirc) &= \perp \\
\delta(q_2, \bigstar) &= (q_2, q_2) \\
\delta(q_3, \bigcirc) &= (q_2, q_2) \\
\delta(q_3, \neg \bigcirc) &= (q_3, q_3)
\end{align*}
\]

Verifying CTL properties – automata-theoretic approach

Alternating tree automata

\[ \delta(q_0, \cdot) = [(q_1, q_2) \lor (q_2, q_1)] \land (q_3, q_3) \]

\[ \delta(q_0, \lnot \cdot) = \bot \]

\[ \delta(q_1, \cdot) = (q_1, q_2) \lor (q_2, q_1) \]

\[ \delta(q_1, \lnot \cdot) = \bot \]

\[ \delta(q_2, \star) = (q_2, q_2) \]

\[ \delta(q_3, \cdot) = (q_2, q_2) \]

\[ \delta(q_3, \lnot \cdot) = (q_3, q_3) \]

This automaton corresponds to \( \text{E} \ G \cdot \land \text{A} \ F \cdot \)

Detailed explanations during François’ lectures.

Behavioural equivalence for CTL

When are two Kripke structures indistinguishable by CTL?

Example

\( \text{EX} \land \text{EX} \)
Behavioural equivalence for CTL

When are two Kripke structures indistinguishable by CTL?

Example
Behavioural equivalence for CTL

When are two Kripke structures indistinguishable by CTL?

Example

\[ \text{EX(\text{EX} \circ \land \text{EX} \circ)} \]
Definition

A relation $B \subseteq Q \times Q$ is a **bisimulation** if
Behavioural equivalence for CTL

Definition

A relation $B \subseteq Q \times Q$ is a bisimulation if

\[ \begin{align*}
\text{same label;} \\
\text{successors on one side have corresponding successors on other side.}
\end{align*} \]
A relation $\mathcal{B} \subseteq Q \times Q$ is a bisimulation if

- same label;
Definition

A relation $B \subseteq Q \times Q$ is a **bisimulation** if

- same label;
- successors on one side have corresponding successors on other side.
A relation $B \subseteq Q \times Q$ is a **bisimulation** if

- **same label**;
- **successors** on one side have corresponding **successors** on other side.
Behavioural equivalence for CTL

**Definition**

A relation $\mathcal{B} \subseteq Q \times Q$ is a **bisimulation** if

- same label;
- successors on one side have **corresponding successors** on other side.
Behavioural equivalence for CTL

Definition

A relation $\mathcal{B} \subseteq Q \times Q$ is a bisimulation if

- same label;
- successors on one side have corresponding successors on other side.
Behavioural equivalence for CTL

Definition

A relation $B \subseteq Q \times Q$ is a bisimulation if

- same label;
- successors on one side have corresponding successors on other side.
Behavioural equivalence for CTL

Theorem ([BCG88])

(The initial states of) two finitely branching Kripke structures satisfy the same CTL formulas if, and only if, they are bisimilar.

Proof
bisimilar states have the same possible futures, hence satisfy same CTL formulas (by induction); non-bisimilar states with finite branching can be distinguished (with only \( \text{EX} \) and \( \text{AX} \)).

Theorem ([BCG88])

(The initial states of) two finitely branching Kripke structures satisfy the same CTL formulas if, and only if, they are bisimilar.

Proof

Behavioural equivalence for CTL

**Theorem ([BCG88])**

(The initial states of) two *finitely branching* Kripke structures satisfy the same CTL formulas if, and only if, they are bisimilar.

**Proof**

- bisimilar states have the same possible futures, hence satisfy same CTL formulas (by induction);

Behavioural equivalence for CTL

Theorem ([BCG88])

(The initial states of) two finitely branching Kripke structures satisfy the same CTL formulas if, and only if, they are bisimilar.

Proof

- bisimilar states have the same possible futures, hence satisfy same CTL formulas (by induction);

- non-bisimilar states with finite branching can be distinguished (with only $\exists X$ and $\forall X$).

Theorem ([BCG88])

(The initial states of) two finitely branching Kripke structures satisfy the same CTL formulas if, and only if, they are bisimilar.

Counter-example

Computation-Tree Logic (CTL*)
Computation-Tree Logic (CTL*)

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

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

- **temporal modalities:**
  - $X \varphi$  
  - $\varphi \mathop{U} \psi$  

- **path quantifiers:**
  - $E \varphi$
  - $A \varphi$
Computation-Tree Logic (CTL*)

\[ \text{any request is eventually served along any fair execution} \]

\[ \mathbf{A} \mathbf{G}(\text{req}_2) \Rightarrow \mathbf{A}(\mathbf{G} \mathbf{F} \text{serv}_i \Rightarrow \mathbf{F} \text{serv}_2) \]
Computation-Tree Logic (CTL*)

\[ \text{AG}(\text{req}_2) \Rightarrow \text{A(GF} \text{serv}_i \Rightarrow \text{F} \text{serv}_2) \]

any request is eventually served along any fair execution

Expressiveness

- CTL* subsumes LTL and CTL;
- two states are bisimilar iff they satisfy the same CTL* formulas.
Computation-Tree Logic (CTL*)

\[
\text{AG(req}_2\Rightarrow\text{AG F serv}_i\Rightarrow\text{F serv}_2) \quad \text{any request is eventually served along any fair execution}
\]

**Expressiveness**

- CTL* subsumes LTL and CTL;
- two states are bisimilar iff they satisfy the same CTL* formulas.

**Theorem ([ES84])**

CTL* model checking is PSPACE-complete.

Outline of the presentation

1. Introduction

2. Basics of temporal logics
   - convenient formalism for expressing properties of reactive systems
   - efficient verification algorithms for finite-state models

3. Temporal logics for multi-agent systems
   - reasoning about interacting components in complex systems
   - temporal logics for expressing controllability properties
Reasoning about open systems

Example (An elevator)

Non-determinism due to interactions with users; underspecified controller for the lift.
Reasoning about open systems

Example (An elevator)

Non-determinism due to interactions with users; underspecified controller for the lift.
Reasoning about open systems

Example (An elevator)

Non-determinism due to
- interactions with users;
Reasoning about open systems

Example (An elevator)

Non-determinism due to
- interactions with users;
- underspecified controller for the lift.
Reasoning about open systems

Example (An elevator)

Controllability properties:

does there exist a controller under which the system satisfies some property, whatever the users do?
Reasoning about open systems
Reasoning about open systems

Games on graphs

A concurrent game structure is made of
- a Kripke structure
A concurrent game structure is made of

- a Kripke structure
- a set of agents (or players);
Reasoning about open systems

**Games on graphs**

A concurrent game structure is made of
- a Kripke structure
- a set of agents (or players);
- a table indicating the transition to be taken given the actions of the players.
Reasoning about open systems

Games on graphs
A concurrent game structure is made of
- a Kripke structure
- 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 structure is a game where each state is controlled by a single player.
Reasoning about open systems

Example (An elevator)

Controllability properties:

does there exist a controller under which the system satisfies some property, whatever the users do?
Reasoning about open systems

Example (An elevator)

Controllability properties:

does there exist a controller under which the system satisfies some property, whatever the users do?
Reasoning about open systems

Example (An elevator)

Controllability properties:

is there a strategy for the circle player to enforce some property against any strategy of the square player?
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.
Reasoning about open systems

**Strategies**

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

**Example**

*Strategy for player*
Reasoning about open systems

**Strategies**

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

**Example**

Strategy for player

alternately go to \( \bullet \) and \( \bigcirc \)

(starting with \( \bigcirc \)).
Reasoning about open systems

**Strategies**

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

**Example**

Strategy for player

alternately go to \( \bigcirc \) and \( \bigcirc \) (starting with \( \bigcirc \)).
Reasoning about open systems

**Strategies**

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

**Example**

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

**Strategies**

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

**Example**

Strategy for player

alternately go to \( \bigcirc \) and \( \bigotimes \) (starting with \( \bigcirc \)).
Reasoning about open systems

**Strategies**

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

**Example**

Memoryless strategy for player □
Reasoning about open systems

**Strategies**

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

**Example**

Memoryless strategy for player □
always go to ○.
Strategies

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

Example

Memoryless strategy for player □
always go to ○.
Reasoning about open systems

**Strategies**

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

**Example**

Memoryless strategy for player always go to .
Reasoning about open systems

Strategies

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

Example

Memoryless strategy for player □
always go to □.
Temporal logics for games: ATL [AHK02]

ATL extends CTL with strategy quantifiers

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

Temporal logics for games: ATL [AHK02]

ATL extends CTL with strategy quantifiers

⟨⟨ A ⟩⟩ ϕ expresses that A has a strategy to enforce ϕ.

⟨◯⟩ F floor₁

there is a controller with which the cabin eventually reaches the first floor (whatever Player □ does).

Temporal logics for games: ATL [AHK02]

ATL extends CTL with **strategy quantifiers**

\[ \langle A \rangle \varphi \] expresses that \( A \) has a strategy to enforce \( \varphi \). 

\[ \langle \Diamond \rangle \mathbf{F} \text{floor}_1 \]  

*there is a controller with which the cabin eventually reaches the first floor (whatever Player \( \Box \) does).*

\[ \langle \Diamond \rangle \mathbf{G}(\text{op}_1 \Rightarrow \text{serv}_1) \]  

*there is a controller for which, if the door is open, the cabin is present.*

Temporal logics for games: ATL [AHK02]

ATL extends CTL with strategy quantifiers

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

\[
\begin{align*}
\langle \bigcirc \rangle \ F \text{floor}_1 & \quad \text{there is a controller with which the cabin eventually reaches the first floor (whatever Player } \Box \text{ does).} \\
\langle \bigcirc \rangle \ G(\text{op}_1 \Rightarrow \text{serv}_1) & \quad \text{there is a controller for which, if the door is open, the cabin is present.} \\
A \ G(\text{req}_0 \Rightarrow \langle \bigcirc \rangle \ F \text{serv}_0) & \quad \text{any request can be granted by the controller}
\end{align*}
\]

Expressiveness of ATL

Semantics of $\langle A \rangle \varphi$

Existential quantification (over strategies) implicitly includes a universal quantification (over outcomes):

$G, \bigcirc \models \langle A \rangle \varphi \iff \exists \sigma_A. \forall \pi \in \text{Out}(\bigcirc, \sigma_A). \pi \models \varphi.$

Proposition $E$ and $A$ can be expressed using $\langle \cdot \rangle$.

Proof

$A \varphi \equiv \langle \emptyset \rangle \varphi$

$E \varphi \equiv \neg \langle \emptyset \rangle \neg \varphi$

Remark

Writing $E \varphi \equiv \langle A_{gt} \rangle \varphi$ would depend on the set of agents...
Expressiveness of ATL

**Semantics of $\langle A \rangle \varphi$**

Existential quantification (over strategies) implicitly includes a universal quantification (over outcomes):

$$G, \Box \models \langle A \rangle \varphi \iff \exists \sigma_A. \forall \pi \in \text{Out}(\Box, \sigma_A). \pi \models \varphi.$$

**Proposition**

$E$ and $A$ can be expressed using $\langle \cdot \rangle$.
Expressiveness of ATL

**Semantics of $\langle A \rangle \varphi$**

Existential quantification (over strategies) implicitly includes a universal quantification (over outcomes):

$$G, \bigcirc \models \langle A \rangle \varphi \iff \exists \sigma_A. \forall \pi \in \text{Out}(\bigcirc, \sigma_A). \pi \models \varphi.$$

**Proposition**

$E$ and $A$ can be expressed using $\langle \cdot \rangle$.

**Proof**

$$A\varphi \equiv \langle \emptyset \rangle \varphi \quad \quad \quad E\varphi \equiv \neg \langle \emptyset \rangle \neg \varphi$$
## Expressiveness of ATL

### Semantics of $\langle A \rangle \varphi$

Existential quantification (over strategies) implicitly includes a universal quantification (over outcomes):

$$G, \bigcirc \models \langle A \rangle \varphi \iff \exists \sigma_A. \forall \pi \in \text{Out}(\bigcirc, \sigma_A). \pi \models \varphi.$$ 

### Proposition

$E$ and $A$ can be expressed using $\langle \cdot \rangle$.

### Proof

$A\varphi \equiv \langle \emptyset \rangle \varphi$ \hspace{1cm} $E\varphi \equiv \neg \langle \emptyset \rangle \neg \varphi$

### Remark

Writing $E\varphi \equiv \langle \text{Agt} \rangle \varphi$ would depend on the set of agents...
Expressiveness of ATL

Definition

\[ [A] \varphi \equiv \neg \langle A \rangle \neg \varphi \]
Expressiveness of ATL

Definition

\[[A] \varphi \equiv \neg \langle A \rangle \neg \varphi\]

Example

\[[A] \text{ F }\bigcirc \]
Expressiveness of ATL

Definition

\[[A] \varphi \equiv \neg \langle A \rangle \neg \varphi\]

Example

\[[A] F \bigcirc \quad \text{for any strategy of } A, \text{ there is an outcome visiting } \bigcirc\]
Expressiveness of ATL

Theorem

\[ \langle A \rangle X, \langle A \rangle G \text{ and } \langle A \rangle U \text{ are not sufficient to express ATL.} \]
Expressiveness of ATL

Theorem

\[ \langle A \rangle X, \langle A \rangle G \text{ and } \langle A \rangle U \text{ are not sufficient to express ATL.} \]

Example

Relaxed version of \textit{until} in CTL:

\[ E \circ W \equiv E \left[ U \lor G \right] \]
Expressiveness of ATL

Theorem

\(\langle A \rangle X, \langle A \rangle G\) and \(\langle A \rangle U\) are not sufficient to express ATL.

Example

Relaxed version of \textit{until} in CTL:

\[
\begin{align*}
E \bigcirc W & \equiv E [U \bigcirc \lor G \bigcirc] \\
& \equiv (E \bigcirc U \bigcirc) \lor (E G \bigcirc)
\end{align*}
\]
Expressiveness of ATL

**Theorem**

\[ \langle A \rangle X, \langle A \rangle G \text{ and } \langle A \rangle U \text{ are not sufficient to express ATL.} \]

**Example**

Relaxed version of \textit{until} in CTL:

\[
E \bigcirc W \equiv E [U \bigcirc G]
\]

\[
\equiv (E \bigcirc U) \lor (E G)
\]

In ATL (actually ATL\(^*\)):

\[ \langle A \rangle \bigcirc W \equiv \langle A \rangle [U \bigcirc G] \]
Expressiveness of ATL

Theorem

\(\langle A \rangle X, \langle A \rangle G \text{ and } \langle A \rangle U\) are not sufficient to express ATL.

Example

Relaxed version of until in CTL:

\[E \bigcirc W \equiv E [U \lor G]\]
\[\equiv (E \bigcirc U) \lor (E G)\]

In ATL (actually ATL\(^*\)):

\[\langle A \rangle \bigcirc W \equiv \langle A \rangle [U \lor G]\]
\[\not\equiv \langle A \rangle \bigcirc U \lor \langle A \rangle G\]
Expressiveness of ATL

Theorem

\( \langle A \rangle X, \langle A \rangle G \text{ and } \langle A \rangle U \) are not sufficient to express ATL.

Example

Relaxed version of until in CTL:

\[
E \odot W \equiv E [ U \lor G ] \\
\equiv (E \odot U) \lor (E G)
\]

In ATL (actually ATL\(^*\)):

\[
\langle A \rangle \odot W \equiv \langle A \rangle [ U \lor G ] \\
\neq \langle A \rangle \odot U \lor \langle A \rangle G
\]

player B

player A
Expressiveness of ATL

**Theorem**

$\langle A \rangle X$, $\langle A \rangle G$ and $\langle A \rangle U$ are not sufficient to express ATL.

**Proof**
Expressiveness of ATL

**Theorem**

\[ \langle A \rangle X, \langle A \rangle G \text{ and } \langle A \rangle U \text{ are not sufficient to express ATL.} \]

**Proof**

\[ s_i \not\models \langle A \rangle W \text{ but } s_i' \models \langle A \rangle W \]
Expressiveness of ATL

Theorem

\[ \langle A \rangle X, \langle A \rangle G \text{ and } \langle A \rangle U \text{ are not sufficient to express ATL.} \]

Remark

Turn-based games are determined:

\[ \neg (\langle A \rangle \varphi) \equiv \langle \text{Agt} \setminus A \rangle \neg \varphi. \]
Expressiveness of ATL

Theorem

\[ \langle A \rangle X, \langle A \rangle G \text{ and } \langle A \rangle U \text{ are not sufficient to express ATL.} \]

Remark

Turn-based games are determined:

\[ \neg (\langle A \rangle \varphi) \equiv \langle \text{Agt} \setminus A \rangle \neg \varphi. \]

Then:

\[ \langle A \rangle \circ \text{W} \circ \equiv \langle A \rangle \neg (\neg \circ U (\neg \circ \wedge \neg \circ)) \]
Expressiveness of ATL

Theorem

\[ \langle A \rangle X, \langle A \rangle G \text{ and } \langle A \rangle U \text{ are not sufficient to express ATL.} \]

Remark

Turn-based games are determined:

\[ \neg (\langle A \rangle \varphi) \equiv \langle \text{Agt} \setminus A \rangle \neg \varphi. \]

Then:

\[ \langle A \rangle \blacklozenge W \blacklozenge \equiv \langle A \rangle \neg (\neg \blacklozenge \bigcirc \ U \bigcirc \neg \bigcirc \land \neg \bigcirc) \]

\[ \equiv_{t.b.} \neg \langle \text{Agt} \setminus A \rangle (\neg \bigcirc) \ U (\neg \bigcirc \land \neg \bigcirc) \]
Model checking ATL [AHK02]

Theorem ([AHK02])

Model checking ATL over turn-based games is PTIME-complete.

Model checking ATL [AHK02]

Theorem ([AHK02])
Model checking ATL over turn-based games is $\text{PTIME}$-complete.

Model checking ATL [AHK02]

Theorem ([AHK02]) Model checking ATL over turn-based games is PTIME-complete.

Model checking ATL [AHK02]

Model checking ATL [AHK02]

Theorem ([AHK02]) Model checking ATL over turn-based games is $\text{PTIME}$-complete.

Model checking ATL [AHK02]

Theorem ([AHK02])

Model checking ATL over turn-based games is PTIME-complete.

Model checking ATL [AHK02]

Theorem ([AHK02]) Model checking ATL over turn-based games is PTIME-complete.

Model checking ATL [AHK02]

Theorem ([AHK02])

Model checking ATL over turn-based games is \( \text{PTIME} \)-complete.

Model checking ATL [AHK02]

Theorem ([AHK02])

Model checking ATL over turn-based games is PTIME-complete.

Model checking ATL [AHK02]

Theorem ([AHK02])

Model checking ATL over turn-based games is PTIME-complete.

Model checking ATL \[\text{[AHK02]}\]

\begin{itemize}
  \item \(\langle\bigcirc\rangle F\)
  \item \(\langle\blacksquare\rangle F\)
  \item \(\langle\bigcirc\rangle G(\langle\blacksquare\rangle F)\)
\end{itemize}

\[\text{Theorem (AHK02)}\]
Model checking ATL over turn-based games is PTIME-complete.

Theorem ([AHK02])

Model checking ATL over turn-based games is PTIME-complete.

Model checking ATL [AHK02]

\begin{align*}
\langle \lozenge \rangle F & \equiv \langle \Box \rangle \big( \langle \lozenge \rangle F \big) \\
G p & \equiv \langle \lozenge \rangle G p
\end{align*}

Model checking ATL [AHK02]

Theorem ([AHK02])

Model checking ATL over turn-based games is PTIME-complete.

Model checking ATL [AHK02]

Theorem ([AHK02])

Model checking ATL over turn-based games is PTIME-complete.

Model checking ATL [AHK02]

Theorem ([AHK02]) Model checking ATL over concurrent games is PTIME-complete.

Model checking ATL \[\text{[AHK02]}\]

Theorem (\[\text{[AHK02]}\]) Model checking ATL over concurrent games is PTIME-complete.

Model checking ATL [AHK02]

Theorem ([AHK02])

Model checking ATL over concurrent games is PTIME-complete.

Model checking ATL [AHK02]

Theorem ([AHK02]) Model checking ATL over concurrent games is PTIME-complete.

Model checking ATL \cite{AHK02}

Theorem (\cite{AHK02}) Model checking ATL over concurrent games is \textbf{PSPACE}-complete.

\cite{AHK02} Alur, Henzinger, Kupferman. Alternating-time Temporal Logic. J. ACM, 2002.
Theorem ([AHK02])

Model checking ATL over concurrent games is \( \text{PTIME} \)-complete.

Model checking ATL [AHK02]

\[ \langle 1.1 \rangle, \langle 2.2 \rangle, \langle 3.3 \rangle \]

\[ \langle 1.2 \rangle, \langle 1.3 \rangle, \langle 3.2 \rangle \]

\[ \langle 2.1 \rangle, \langle 2.3 \rangle, \langle 3.1 \rangle \]

\[ \langle 1 \rangle \quad F \]

Player 1

Player 2

\[ \begin{array}{ccc}
\mathcal{A} & \mathcal{A} & 3 \\
1 & & \\
2 & & \\
3 & & \\
\end{array} \]

Model checking ATL [AHK02]

Theorem ([AHK02]) Model checking ATL over concurrent games is PTIME-complete.

Model checking ATL [AHK02]

Theorem ([AHK02]) Model checking ATL over concurrent games is PTIME-complete.

Model checking ATL [AHK02]

Theorem ([AHK02])

*Model checking ATL over concurrent games is PTIME-complete.*

Model checking ATL [AHK02]

\[
\langle 1.1 \rangle, \langle 2.2 \rangle, \langle 3.3 \rangle
\]
\[
\langle 1.2 \rangle, \langle 1.3 \rangle, \langle 3.2 \rangle
\]
\[
\langle 2.1 \rangle, \langle 2.3 \rangle, \langle 3.1 \rangle
\]

\[\text{if } m_1 = m_2 \text{ goto } \]
\[\text{otherwise, if } m_1 = 1 \text{ or } m_2 = 2 \text{ goto } \]
\[\text{otherwise, goto } \]
\[\langle\langle 1 \rangle\rangle \]
\[F \]

Theorem ([LMO07])
Model checking ATL over concurrent games with symbolic transition table is \(\Delta P^3\)-complete (hence in \(PSPACE\), \(NP\)-hard).


Model checking ATL \[\text{[AHK02]}\]

\[
\langle 1.1\rangle, \langle 2.2\rangle, \langle 3.3\rangle
\]

\[
\langle 1.2\rangle, \langle 1.3\rangle, \langle 3.2\rangle
\]

\[
\langle 2.1\rangle, \langle 2.3\rangle, \langle 3.1\rangle
\]

- if \(m_1 = m_2\) goto ●;
- otherwise, if \(m_1 = 1\) or \(m_2 = 2\) goto ○;
- otherwise, goto ○;

\[\langle\langle 1.1\rangle\rangle F\]

for each move of Player 1, need to solve several instances of SAT

Theorem (\[\text{[LMO07]}\])

Model checking ATL over concurrent games with symbolic transition table is \(\Delta_3\)-complete (hence in \(\text{PSPACE}\), \(\text{NP}\)-hard).


Model checking ATL [AHK02]

- if $m_1 = m_2$ goto $\bullet$;
- otherwise, if $m_1 = 1$ or $m_2 = 2$ goto $\circ$;
- otherwise, goto $\circ$;

$\langle\langle 1 \rangle\rangle F$
Model checking ATL \[\text{[AHK02]}\]

\[\langle 1.1, 2.2, 3.3 \rangle, \langle 2.1, 2.3, 3.1 \rangle, \langle 1.2, 1.3, 3.2 \rangle \]

- if \( m_1 = m_2 \) goto \( \bullet \);
- otherwise, if \( m_1 = 1 \) or \( m_2 = 2 \) goto \( \bullet \);
- otherwise, goto \( \bullet \);

\( \langle 1 \rangle \) F \( \bullet \)


Model checking ATL [AHK02]

\[\begin{align*}
\langle 1.1, 2.2, 3.3 \rangle & \quad \text{if } m_1 = m_2 \text{ goto } \bigcirc; \\
\langle 1.2, 1.3, 3.2 \rangle & \quad \text{otherwise, if } m_1 = 1 \text{ or } m_2 = 2 \text{ goto } \bigcirc; \\
\langle 2.1, 2.3, 3.1 \rangle & \quad \text{otherwise, goto } \bigcirc; \\
\end{align*}\]

\[\langle 1 \rangle \Box \quad \text{F } \bigcirc\]

\[\sim \text{ for each move of Player 1, need to solve several instances of SAT}\]


Model checking ATL [AHK02]

if $m_1 = m_2$ goto ●;
otherwise, if $m_1 = 1$ or $m_2 = 2$ goto ○;
otherwise, goto ○;

$\langle 1.1, 2.2, 3.3 \rangle$

$\langle 1.2, 1.3, 3.2 \rangle$

$\langle 2.1, 2.3, 3.1 \rangle$

$\langle 1 \rangle$ $\text{F}$ ○

$\leadsto$ for each move of Player 1, need to solve several instances of SAT

Theorem ([LMO07])

Model checking ATL over concurrent games with symbolic transition table is $\Delta_3^P$-complete (hence in PSPACE, NP-hard).

The full logic ATL*
The full logic ATL*

\[ \langle A \rangle \left[ GF \text{serv}_i \Rightarrow G(\text{req}_0 \Rightarrow F \text{serv}_0) \right] \]
The full logic ATL$^*$

$\langle A \rangle \left[ GF \text{serv}_i \Rightarrow GF\text{req}_0 \Rightarrow F\text{serv}_0 \right]$  

**Theorem ([AHK02])**

*Model checking ATL$^*$ is 2EXPTIME-complete.*

The full logic ATL*

\[ \langle A \rangle \left[ GF\text{ serv}_i \Rightarrow G(\text{ req}_0 \Rightarrow F\text{ serv}_0) \right] \]

**Theorem ([AHK02])**

*Model checking ATL* is 2EXPTIME-complete.

**Proof**

Games with LTL objectives can be solved in 2EXPTIME:

- LTL formula to deterministic parity automaton:
  - size doubly-exponential;
  - number of priorities exponential.
- solve parity game in \( O(|Q||p|) \).

Behavioural equivalence for ATL

A relation \( B \subseteq Q \times Q \) is an alternating bisimulation if:

- same label;
- any action on one side has a corresponding action on other side.
Definition

A relation $\mathcal{B} \subseteq Q \times Q$ is an alternating bisimulation if

- same label;
- any action on one side has a corresponding action on other side.
Behavioural equivalence for ATL

**Definition**

A relation \( B \subseteq Q \times Q \) is an **alternating bisimulation** if

- same label;
Behavioural equivalence for ATL

Definition

A relation $\mathcal{B} \subseteq Q \times Q$ is an alternating bisimulation if

- same label;
- any action on one side has a corresponding action on other side.
Behavioural equivalence for ATL

Definition

A relation $\mathcal{B} \subseteq Q \times Q$ is an alternating bisimulation if:

- same label;
- any action on one side has a corresponding action on other side.
Behavioural equivalence for ATL

**Definition**

A relation $\mathcal{B} \subseteq Q \times Q$ is an **alternating bisimulation** if

- same label;
- any action on one side has a **corresponding action** on other side.
Behavioural equivalence for ATL

**Definition**

A relation $\mathcal{B} \subseteq Q \times Q$ is an **alternating bisimulation** if

- same label;
- any action on one side has a corresponding action on other side.
A relation $B \subseteq Q \times Q$ is an alternating bisimulation if

- same label;
- any action on one side has a corresponding action on other side.
A relation $B \subseteq Q \times Q$ is an alternating bisimulation if

- same label;
- any action on one side has a corresponding action on other side.
Behavioural equivalence for ATL

**Definition**

A relation $B \subseteq Q \times Q$ is an alternating bisimulation if

- same label;
- any action on one side has a corresponding action on the other side.
Behavioural equivalence for ATL

**Definition**

A relation $B \subseteq Q \times Q$ is an alternating bisimulation if

- same label;
- any action on one side has a corresponding action on the other side.

**Theorem ([AHKV98])**

(The initial states of) two concurrent game structures (with finitely many moves) satisfy the same ATL formulas if, and only if, they are alternating-bisimilar.

Example

\[\langle \square \rangle \text{G} \langle \square \rangle \text{F} \text{O} \]

Brihaye, Da Costa, Laroussinie, Markey. ATL with strategy contexts and bounded memory. LFCS, 2009.
Da Costa, Laroussinie, Markey. ATL with strategy contexts: expressiveness and ... FSTTCS, 2010.
ATL with strategy contexts [BDLM09,DLM10]

Example

\[
\langle \Box \rangle G \langle \square \rangle F \langle \Diamond \rangle
\]
ATL with strategy contexts [BDLM09,DLM10]

Example

Player $\Diamond$ in $\Box$ always plays to $\square$.

Brihaye, Da Costa, Laroussinie, Markey. ATL with strategy contexts and bounded memory. LFCS, 2009.
Da Costa, Laroussinie, Markey. ATL with strategy contexts: expressiveness and ... FSTTCS, 2010.
ATL with strategy contexts [BDLM09, DLM10]

Example

- Player $\bigcirc$ in $\bigcirc$ always plays to $\square$.

Brihaye, Da Costa, Laroussinie, Markey. ATL with strategy contexts and bounded memory. LFCS, 2009.
Da Costa, Laroussinie, Markey. ATL with strategy contexts: expressiveness and ... FSTTCS, 2010.
ATL with strategy contexts [BDLM09, DLM10]

Example

Player $\bigcirc$ in $\bigcirc$ always plays to $\square$.

Brihaye, Da Costa, Laroussinie, Markey. ATL with strategy contexts and bounded memory. LFCS, 2009.
Da Costa, Laroussinie, Markey. ATL with strategy contexts: expressiveness and ... FSTTCS, 2010.
Example

- Player $\bigcirc$ in $\bigcirc$ always plays to $\Box$;
- Player $\Box$ in $\square$ then plays to $\bigcirc$.

Brihaye, Da Costa, Laroussinie, Markey. ATL with strategy contexts and bounded memory. LFCS, 2009.
Da Costa, Laroussinie, Markey. ATL with strategy contexts: expressiveness and ... FSTTCS, 2010.
ATL with strategy contexts

Definition

\( \text{ATL}_{sc} \) has several new strategy quantifiers:

\[ \langle \cdot A \cdot \rangle \text{ is similar to } \langle \langle A \rangle \rangle \text{ but assigns the corresponding strategy to } A \text{ for evaluating } \phi; \]
\[ \langle \cdot A \cdot \rangle \equiv \langle \cdot A_{gt} \setminus A \cdot \rangle, \text{ which will be useful to get formulas that do not depend on the underlying set of agents}; \]
\[ \langle \cdot A \cdot \rangle_0 \text{ is similar to } \langle \cdot A \cdot \rangle \text{ but quantifies over memoryless strategies}; \]
\[ \lbrack \cdot A \cdot \rbrack \text{ is dual to } \langle \cdot A \cdot \rangle : \]
\[ \lbrack \cdot A \cdot \rbrack \phi \equiv \neg \langle \cdot A \cdot \rangle \neg \phi \]
ATL with strategy contexts

Definition

$\text{ATL}_{sc}$ has several new strategy quantifiers:

- $\langle \cdot A \cdot \rangle$ is similar to $\langle A \rangle$ but assigns the corresponding strategy to $A$ for evaluating $\varphi$;

\[ \langle \cdot A \cdot \rangle \equiv \langle \cdot A_{\text{gt}} \cdot \rangle \text{, which will be useful to get formulas that do not depend on the underlying set of agents;} \]

\[ \langle \cdot A \cdot \rangle_0 \text{ is similar to } \langle \cdot A \cdot \rangle \text{ but quantifies over memoryless strategies;} \]

\[ [\cdot A \cdot] \text{ is dual to } \langle \cdot A \cdot \rangle : [\cdot A \cdot] \varphi \equiv \neg \langle \cdot A \cdot \rangle \neg \varphi \]
ATL with strategy contexts

Definition

$\text{ATL}_{sc}$ has several new strategy quantifiers:

- $\langle \cdot A \cdot \rangle$ is similar to $\langle A \rangle$ but assigns the corresponding strategy to $A$ for evaluating $\varphi$;
- $\langle \cdot \overline{A} \cdot \rangle \equiv \langle \text{Agt} \setminus A \rangle$, which will be useful to get formulas that do not depend on the underlying set of agents;
ATL with strategy contexts

Definition

\( \text{ATL}_{sc} \) has several \textbf{new strategy quantifiers}:

- \( \langle \cdot \ A \cdot \rangle \) is similar to \( \langle A \rangle \) but assigns the corresponding strategy to \( A \) for evaluating \( \varphi \);

- \( \langle \cdot \overline{A} \cdot \rangle \equiv \langle \text{Agt} \setminus A \rangle \), which will be useful to get formulas that do not depend on the underlying set of agents;

- \( \langle A \rangle_0 \) is similar to \( \langle A \rangle \) but quantifies over \textbf{memoryless} strategies;
ATL with strategy contexts

Definition

$\text{ATL}_{sc}$ has several new strategy quantifiers:

- $\langle \cdot A \cdot \rangle$ is similar to $\langle A \rangle$ but assigns the corresponding strategy to $A$ for evaluating $\varphi$;
- $\langle \cdot \overline{A} \cdot \rangle \equiv \langle \cdot \text{Agt} \setminus A \cdot \rangle$, which will be useful to get formulas that do not depend on the underlying set of agents;
- $\langle \cdot A \cdot \rangle_0$ is similar to $\langle \cdot A \cdot \rangle$ but quantifies over memoryless strategies;
- $\langle A \rangle$ drops the assigned strategies for $A$;
ATL with strategy contexts

Definition

ATL\(_{sc}\) has several **new strategy quantifiers**:

- \(\langle \cdot A \cdot \rangle\) is similar to \(\langle A \rangle\) but **assigns** the corresponding strategy to \(A\) for evaluating \(\varphi\);

- \(\langle \cdot \overline{A} \cdot \rangle \equiv \langle \text{Agt} \setminus A \rangle\), which will be useful to get formulas that do not depend on the underlying set of agents;

- \(\langle \cdot A \cdot \rangle_0\) is similar to \(\langle \cdot A \cdot \rangle\) but quantifies over **memoryless** strategies;

- \(\langle A \rangle\) **drops** the assigned strategies for \(A\);

- \([A]\) is **dual** to \(\langle \cdot A \cdot \rangle\):

\[
[A] \varphi \equiv \neg \langle \cdot A \cdot \rangle \neg \varphi
\]
Semantics of $\langle \cdot A \cdot \rangle \varphi$

**Definition**

Semantics of ATL strategy quantifier:

$\mathcal{G}, \bigcirc \models \langle A \rangle \varphi \iff \exists \sigma_A. \forall \pi \in \text{Out}(\bigcirc, \sigma_A). \pi \models \varphi$
Semantics of $\langle \cdot A \rangle \varphi$

**Definition**

**Semantics of ATL strategy quantifier:**

$\mathcal{G}, \emptyset \models \langle A \rangle \varphi \iff \exists \sigma_A. \forall \pi \in \text{Out}(\emptyset, \sigma_A). \pi \models \varphi$

**Semantics of ATL$_{sc}$ strategy quantifier:**

$\mathcal{G}, \emptyset \models_{\sigma_B} \langle A \rangle \varphi \iff \exists \sigma_A. \forall \pi \in \text{Out}(\emptyset, \sigma_A \circ \sigma_B). \pi \models_{\sigma_A \circ \sigma_B} \varphi$
Semantics of $\langle \cdot A \rangle \varphi$

**Definition**

Semantics of ATL strategy quantifier:

$\mathcal{G}, \bigcirc \models \langle A \rangle \varphi \iff \exists \sigma_A. \forall \pi \in \text{Out}(\bigcirc, \sigma_A). \pi \models \varphi$

Semantics of $\text{ATL}_{sc}$ strategy quantifier:

$\mathcal{G}, \bigcirc \models_{\sigma_B} \langle \cdot A \rangle \varphi \iff \exists \sigma_A. \forall \pi \in \text{Out}(\bigcirc, \sigma_A \circ \sigma_B). \pi \models_{\sigma_A \circ \sigma_B} \varphi$

- **formulas evaluated in a context** (a strategy for some coalition);
- **context initially empty**;
- **newly selected strategies added to the context**:

$$\sigma_A \circ \sigma_B : a \mapsto \sigma_A(a) \text{ if } a \in A \setminus B$$

$$b \mapsto \sigma_B(b) \text{ if } b \in B \setminus A$$

$$c \mapsto \sigma_A(c) \text{ if } c \in B \cap A$$
What $\text{ATL}_{sc}$ can express
What $ATL_{sc}$ can express

- $A$ has a strategy to eventually reach $\bigcirc$: 
  
  $\langle \cdot \cdot \rangle F \equiv \langle \langle A \rangle \rangle F$ in $CTL^*$ and $ATL^*$ properties:
What $\text{ATL}_{sc}$ can express

- $A$ has a strategy to eventually reach $\bigcirc$:
  - $\langle \cdot \rangle F \bigcirc$

memory needed!
What $\text{ATL}_{sc}$ can express

- $A$ has a strategy to eventually reach $\bigcirc$:
  - $\langle A \rangle F \bigcirc$
  - $\langle \overline{C} \rangle \langle A \rangle F \bigcirc$
What \( \text{ATL}_{sc} \) can express

- A has a strategy to eventually reach \( \bigcirc \):
  - \( \langle A \rangle F \bigcirc \)
  - \( (\overline{\square}) \langle A \rangle F \bigcirc \equiv \langle A \rangle F \bigcirc \)
What $\text{ATL}_{sc}$ can express

- $A$ has a strategy to eventually reach $\Diamond$:
  - $\langle \cdot A \rangle \mathbf{F} \Diamond$
  - $\langle \neg \neg \rangle \langle \cdot A \rangle \mathbf{F} \Diamond \equiv \langle A \rangle \mathbf{F} \Diamond$
  - $\langle \cdot A \rangle [\overline{A}] \mathbf{F} \Diamond$

Memory needed!
What $\text{ATL}_\text{sc}$ can express

- $A$ has a strategy to eventually reach $\Diamond$:
  - $\langle \cdot A \cdot \rangle F \Diamond$
  - $\langle \overline{\emptyset} \rangle \langle \cdot A \cdot \rangle F \Diamond \equiv \langle A \rangle F \Diamond$
  - $\langle \cdot A \cdot \rangle [\overline{\cdot A \cdot}] F \Diamond$

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

- $A$ has a strategy to eventually reach $\bigcirc$:
  - $\langle A \rangle \textbf{F} \bigcirc$
  - $\langle \overline{A} \rangle \langle A \rangle \textbf{F} \bigcirc \equiv \langle A \rangle \textbf{F} \bigcirc$
  - $\langle A \rangle \lbrack \overline{A} \rbrack \textbf{F} \bigcirc$

- $\text{CTL}^*$ and $\text{ATL}^*$ properties:
  - $\langle \overline{A} \rangle \textbf{G} \langle \emptyset \rangle \textbf{F} \bigcirc$
What $\text{ATL}_{sc}$ can express

- $\, A$ has a strategy to eventually reach $\bigcirc$:
  - $\langle \cdot A \cdot \rangle \, F \bigcirc$
  - $\langle \overline{\emptyset} \rangle \langle \cdot A \cdot \rangle \, F \bigcirc \equiv \langle A \rangle \, F \bigcirc$
  - $\langle \cdot A \cdot \rangle \, [\overline{A}] \, F \bigcirc$

- $\text{CTL}^*$ and $\text{ATL}^*$ properties:
  - $\langle \overline{\emptyset} \rangle \, G \langle \cdot \emptyset \cdot \rangle \, F \bigcirc \equiv E \, G \, F \bigcirc$
What $\text{ATL}_{sc}$ can express

- $A$ has a strategy to eventually reach $\bullet$:
  - $\langle \cdot A \rangle \ F \bullet$
  - $\langle \cdot A \rangle F \bullet \equiv \langle A \rangle F \bullet$
  - $\langle \cdot A \rangle [\overline{A}] F \bullet$

- $\text{CTL}^*$ and $\text{ATL}^*$ properties:
  - $\langle \cdot \overline{A} \rangle G \langle \cdot \Diamond \rangle F \bullet \equiv E G F \bullet$
  - $\langle \cdot \overline{A} \rangle (G \langle \cdot \Diamond \rangle F \bullet \land G \langle \cdot \Diamond \rangle F \circ)$
What $\text{ATL}_{sc}$ can express

- $A$ has a strategy to eventually reach $\bigcirc$:
  - $\langle \cdot A \cdot \rangle F \bigcirc$
  - $(\overline{\emptyset}) \langle \cdot A \cdot \rangle F \bigcirc \equiv \langle A \rangle F \bigcirc$
  - $\langle A \rangle [\overline{A}] F \bigcirc$

- $\text{CTL}^*$ and $\text{ATL}^*$ properties:
  - $\langle \overline{\emptyset} \rangle G \langle \cdot \emptyset \cdot \rangle F \bigcirc \equiv E G F \bigcirc$
  - $\langle \overline{\emptyset} \rangle (G \langle \cdot \emptyset \cdot \rangle F \bigcirc \land G \langle \cdot \emptyset \cdot \rangle F \bigcirc) \equiv E(G F \bigcirc \land G F \bigcirc)$
What \( \text{ATL}_{sc} \) can express

- \( A \) has a strategy to eventually reach \( \bigcirc \):
  - \( \langle \cdot A \cdot \rangle \text{ F} \bigcirc \)
  - \( (\overline{\varnothing}) \langle \cdot A \cdot \rangle \text{ F} \bigcirc \equiv \langle \langle A \rangle \rangle \text{ F} \bigcirc \)
  - \( \langle A \rangle \overline{[\overline{A}]} \text{ F} \bigcirc \)

- CTL* and ATL* properties:
  - \( \langle \overline{\varnothing} \rangle \text{ G} \langle \cdot \varnothing \cdot \rangle \text{ F} \bigcirc \equiv \text{ E G F} \bigcirc \)
  - \( \langle \overline{\varnothing} \rangle (\text{ G} \langle \cdot \varnothing \cdot \rangle \text{ F} \bigcirc \land \text{ G} \langle \cdot \varnothing \cdot \rangle \text{ F} \bigcirc) \equiv \text{ E(G F} \bigcirc \land \text{ G F} \bigcirc) \)
What $\text{ATL}_{sc}$ can express

- $A$ has a strategy to eventually reach $\bigcirc$:
  - $\langle \cdot A \cdot \rangle F \bigcirc$
  - $\langle \emptyset \rangle \langle \cdot A \cdot \rangle F \bigcirc \equiv \langle A \rangle F \bigcirc$
  - $\langle A \rangle [\overline{A}] F \bigcirc$

- CTL* and ATL* properties:
  - $\langle \emptyset \rangle G \langle \cdot \emptyset \cdot \rangle F \bigcirc \equiv E G F \bigcirc$
  - $\langle \emptyset \rangle (G \langle \cdot \emptyset \cdot \rangle F \bigcirc \land G \langle \cdot \emptyset \cdot \rangle F \bigcirc) \equiv E(G F \bigcirc \land G F \bigcirc)$

$\sim$ memory needed!
What $ATL_{sc}$ can express
What $\text{ATL}_{sc}$ can express

- existence of a correct controller for the elevator:

$$\langle \cdot \text{ctrl} \cdot \rangle \ G \left[ \bigwedge_{i,j} \langle \cdot \text{user} \cdot \rangle_0 \ F \left( \bigwedge \text{op}_i \land F \text{op}_j \right) \right]$$

$$\bigwedge \text{op}_i \Rightarrow \text{serv}_i \right)$$
What $\text{ATL}_{sc}$ can express

- existence of a correct controller for the elevator:
- Client-server interactions for accessing a shared resource:

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

- existence of a **correct controller** for the elevator:

- **Client-server interactions** for accessing a shared resource:

\[
\langle \cdot \text{Server} \cdot \rangle \quad G \quad \bigwedge \limits_{c \in \text{Clients}} \langle \cdot c \cdot \rangle \ F \ \text{access}_c
\]

\[
\neg \bigwedge \limits_{c \neq c'} \text{access}_c \land \text{access}_{c'}
\]

- Existence of **Nash equilibria**:

\[
\langle A_1, \ldots, A_n \rangle \bigwedge \limits_i \left( \langle \cdot A_i \cdot \rangle \varphi_{A_i} \Rightarrow \varphi_{A_i} \right)
\]
What $\text{ATL}_{sc}$ can express

- Existence of a **correct controller** for the elevator:

- **Client-server** interactions for accessing a shared resource:

$$\langle \cdot \text{Server} \cdot \rangle \ G \left[ \bigwedge_{c \in \text{Clients}} \langle \cdot c \cdot \rangle \ F \text{access}_c \right]$$

- Existence of **Nash equilibria**:

$$\langle \cdot A_1, \ldots, A_n \cdot \rangle \bigwedge_{i} \left( \langle \cdot A_i \cdot \rangle \varphi_{A_i} \Rightarrow \varphi_{A_i} \right)$$

- Existence of **dominating strategy**:

$$\langle \cdot A \cdot \rangle \ [B] \left( \neg \varphi \Rightarrow [A] \neg \varphi \right)$$
More expressiveness results

Theorem

\[ ATL_{sc} \text{ is strictly more expressive than } ATL. \]
More expressiveness results

Theorem

\( ATL_{sc} \) is strictly more expressive than \( ATL \).

Proof

\[ \langle A \rangle \varphi \equiv (Agt) \langle A \rangle \hat{\varphi} \]
More expressiveness results

Theorem

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

Proof

- \( \langle A \rangle \psi \equiv \langle \text{Agt} \rangle \langle A \rangle \hat{\psi} \)
- \( \langle 1 \rangle \langle 2 \rangle X a \land \langle 2 \rangle X b \) is true in \( s' \) and not in \( s \).
More expressiveness results

**Theorem**

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

**Proof**

- $\langle A \rangle \varphi \equiv \langle \text{Ag}t \rangle \langle A \rangle \varphi$

- $\langle 1 \rangle (\langle 2 \rangle \mathbf{X} a \land \langle 2 \rangle \mathbf{X} b)$ is true in $s'$ and not in $s$. But $s$ and $s'$ are alternating bisimilar.
More expressiveness results

Theorem

- $ATL_{sc}$ is as expressive as $ATL_{sc}^*$;
- $\langle A \rangle$ does not add expressive power.
More expressiveness results

Theorem

- $\text{ATL}_{sc}$ is as expressive as $\text{ATL}_{sc}^*$;
- $\langle A \rangle$ does not add expressive power.

Proof

- Make implicit quantification explicit:

$$\langle A \rangle \varphi \equiv \langle A \rangle [\text{Agt} \setminus A] \hat{\varphi}$$
More expressiveness results

Theorem

- $\text{ATL}_{sc}$ is as expressive as $\text{ATL}^*_{sc}$;
- $\langle A \rangle$ does not add expressive power.

Proof

- Make implicit quantification explicit:

  \[ \langle \cdot \rangle \varphi \equiv \langle \cdot \rangle [\text{Agt} \setminus A] \hat{\varphi} \]

  - always assume the context is full;
  - keep track of which strategies are really useful.
More expressiveness results

**Theorem**

- $ATL_{sc}$ is as expressive as $ATL_{sc}^*$;
- $\langle A \rangle$ does not add expressive power.

**Proof**

- Make implicit quantification explicit:

  $$\langle \cdot A \cdot \rangle \varphi \equiv \langle \cdot A \cdot \rangle \ [A gt \ \backslash A] \widehat{\varphi}$$

  - always assume the context is full;
  - keep track of which strategies are really useful.

- $\langle A \rangle \varphi$ is then equivalent to $[A] \widehat{\varphi}$;
More expressiveness results

**Theorem**

- $\text{ATL}_{sc}$ is as expressive as $\text{ATL}^*_{sc}$;
- $\langle A \rangle$ does not add expressive power.

**Proof**

- Make implicit quantification explicit:

  $$\langle \cdot A \cdot \rangle \varphi \equiv \langle \cdot A \cdot \rangle [\text{Agt} \setminus A] \hat{\varphi}$$

  - always assume the context is full;
  - keep track of which strategies are really useful.

- $\langle A \rangle \varphi$ is then equivalent to $[A] \hat{\varphi}$;
- for full context, insert $\langle \cdot \emptyset \cdot \rangle$ between temporal modalities:

  $$\text{G F} \varphi \equiv \langle \cdot \emptyset \cdot \rangle \text{G} \langle \cdot \emptyset \cdot \rangle \text{F} \varphi$$
Explicit manipulation of strategies

\[ \exists \sigma. \phi(\sigma) \]

there is a strategy \( \sigma \) such that \( \phi \) holds

assignment of strategies to agents.

\[ \text{assign}(A \mapsto \sigma) \]

if \( A \) plays \( \\sigma \) then \( \phi \) holds

Example

\[ \exists \sigma_1. \text{assign}(A \mapsto \sigma_1) \]
Related works – Strategy logic [CHP07,MMV10]

Explicit manipulation of strategies

- first-order quantification over strategies;

\[ \exists \sigma. \varphi(\sigma) \quad \text{there is a strategy } \sigma \text{ such that } \varphi \text{ holds} \]

- assignment of strategies to agents.

\[ \text{assign}(A \mapsto \sigma)\varphi \quad \text{if } A \text{ plays } \sigma \text{ then } \varphi \text{ holds} \]

Explicit manipulation of strategies

- first-order quantification over strategies;

\[ \exists \sigma. \varphi(\sigma) \quad \text{there is a strategy } \sigma \text{ such that } \varphi \text{ holds} \]

- assignment of strategies to agents.

\[ \text{assign}(A \mapsto \sigma) \varphi \quad \text{if } A \text{ plays } \sigma \text{ then } \varphi \text{ holds} \]

Example

\[ \exists \sigma_1. \text{assign}(A \mapsto \sigma_1). \ \forall \sigma_2. \text{assign}(B \mapsto \sigma_2). \ F \]

Related works – Strategy logic \[\text{[CHP07,MMV10]}\]

Explicit manipulation of strategies

- first-order quantification over strategies;
  \[
  \exists \sigma. \varphi(\sigma) \quad \text{there is a strategy } \sigma \text{ such that } \varphi \text{ holds}
  \]
- assignment of strategies to agents.
  \[
  \text{assign}(A \mapsto \sigma) \varphi \quad \text{if } A \text{ plays } \sigma \text{ then } \varphi \text{ holds}
  \]

Example

\[
\exists \sigma_1. \text{assign}(A \mapsto \sigma_1) \cdot \forall \sigma_2. \text{assign}(B \mapsto \sigma_2) \cdot \ F \cdot \equiv \langle A \rangle F \cdot [B]
\]

Related works – Strategy logic [CHP07,MMV10]

Explicit manipulation of strategies

- first-order quantification over strategies;
  \[ \exists \sigma. \varphi(\sigma) \quad \text{there is a strategy } \sigma \text{ such that } \varphi \text{ holds} \]
- assignment of strategies to agents.
  \[ \text{assign}(A \mapsto \sigma) \varphi \quad \text{if } A \text{ plays } \sigma \text{ then } \varphi \text{ holds} \]

Example

\[ \exists \sigma_1. \text{assign}(A \mapsto \sigma_1). \forall \sigma_2. \text{assign}(B \mapsto \sigma_2). \quad \mathbf{F} \mathbf{0} \equiv \langle \cdot A \cdot \rangle \mathbf{F} \mathbf{0} \]

\[ \exists \sigma_1. \quad \text{AG}(\text{assign}(A \mapsto \sigma_1). \forall \sigma_2. \text{assign}(B \mapsto \sigma_2) \mathbf{F} \mathbf{0}) \]

Related works – Strategy logic [CHP07, MMV10]

Explicit manipulation of strategies

- first-order quantification over strategies;
  \[ \exists \sigma. \varphi(\sigma) \text{ there is a strategy } \sigma \text{ such that } \varphi \text{ holds} \]

- assignment of strategies to agents.
  \[ \text{assign}(A \mapsto \sigma) \varphi \text{ if } A \text{ plays } \sigma \text{ then } \varphi \text{ holds} \]

Proposition

SL is strictly(?) more expressive than ATL_{sc}.

Related works – Strategy logic [CHP07,MMV10]

Explicit manipulation of strategies

- first-order quantification over strategies;

\[ \exists \sigma. \varphi(\sigma) \quad \text{there is a strategy } \sigma \text{ such that } \varphi \text{ holds} \]

- assignment of strategies to agents.

\[ \text{assign}(A \mapsto \sigma) \varphi \quad \text{if } A \text{ plays } \sigma \text{ then } \varphi \text{ holds} \]

Proposition

SL is strictly(?) more expressive than ATL_{sc}.

Theorem

SL model checking is decidable; SL satisfiability is undecidable.

Some other related approaches
Some other related approaches

(Basic) strategy interaction logic [WSH15]

- ATL augmented with strategy interaction quantifiers:

  $\langle \langle A \rangle \rangle (\langle \langle +B \rangle \rangle \varphi_B \land \langle \langle +C \rangle \rangle \varphi_c)$

Some other related approaches

(Basic) strategy interaction logic [WSH15]

- ATL augmented with strategy interaction quantifiers:

\[
\langle A \rangle \left( \langle +B \rangle \varphi_B \land \langle +C \rangle \varphi_c \right)
\]

- context is reset after temporal modalities

Some other related approaches

(Basic) strategy interaction logic [WSH15]

- ATL augmented with strategy interaction quantifiers:
  \[
  \langle A \rangle (\langle +B \rangle \varphi_B \land \langle +C \rangle \varphi_c)
  \]

- context is reset after temporal modalities
- only existential SIQ

Some other related approaches

**Basic) strategy interaction logic** [WSH15]

- ATL augmented with *strategy interaction quantifiers*:
  \[
  \langle\langle A \rangle\rangle (\langle\langle +B \rangle\rangle \varphi_B \land \langle\langle +C \rangle\rangle \varphi_c)
  \]
- **context is reset** after temporal modalities
- **only existential SIQ**

**Proposition ([WSH15])**

*Model checking BSIL is PSPACE-complete.*

Some other related approaches

(Basic) strategy interaction logic [WSH15]

ATL with irrevocable strategies [ÅGJ07]

\[ \langle A \rangle G (\langle A \rangle X \circ) \]

Some other related approaches

(Basic) strategy interaction logic [WSH15]

ATL with irrevocable strategies [ÅGJ07]

- $\langle A \rangle G (\langle A \rangle X O)$

- better use one single strategy per agent;

Some other related approaches

(Basic) strategy interaction logic [WSH15]

ATL with irrevocable strategies [ÅGJ07]

- $⟨⟨A⟩⟩ G (⟨⟨A⟩⟩ X □)$
- better use one single strategy per agent;
- strategies are persistent.

Some other related approaches

(Basic) strategy interaction logic [WSH15]

ATL with irrevocable strategies [ÅGJ07]

- $\langle A \rangle G \langle A \rangle X \Diamond$

- better use one single strategy per agent;

- strategies are persistent.

Proposition

Model checking ATL with irrevocable memoryless strategies is PSPACE-complete.

Some other related approaches

(Basic) strategy interaction logic [WSH15]

ATL with irrevocable strategies [ÅGJ07]

ATL with explicit strategies [WHW07]

\[ \langle A \rangle_{\rho} \varphi: \rho \text{ explicitly imposes a strategy for some players;} \]

Some other related approaches

(Basic) strategy interaction logic [WSH15]

ATL with irrevocable strategies [ÅGJ07]

ATL with explicit strategies [WHW07]

- $\langle A \rangle_\rho \varphi$: $\rho$ explicitly imposes a strategy for some players;

Proposition

*For memoryless explicit strategy contexts, model checking ATLES is PTIME-complete.*

Some other related approaches

(Basic) strategy interaction logic [WSH15]

ATL with irrevocable strategies [ÅGJ07]

ATL with explicit strategies [WHW07]

Stochastic Game Logic [BBGK07]
- extension of ATL\(_{sc}\) to stochastic setting;

(Basic) strategy interaction logic [WSH15]

ATL with irrevocable strategies [ÅGJ07]

ATL with explicit strategies [WHW07]

Stochastic Game Logic [BBGK07]
- extension of ATL$_{sc}$ to stochastic setting;
- quantitative objectives make model checking undecidable;

Some other related approaches

(Basic) strategy interaction logic [WSH15]

ATL with irrevocable strategies [ÅGJ07]

ATL with explicit strategies [WHW07]

Stochastic Game Logic [BBGK07]

- extension of ATL$_{sc}$ to stochastic setting;
- quantitative objectives make model checking undecidable;

Proposition ([BBGK07])

Model checking Stochastic Game Logic with memoryless randomized strategies is decidable in EXPSPACE.