Advanced modelling techniques
Formal verification, temporal logics, model-checking

Laure Petrucci

Université Paris 13
Objectives of the module

- introduce formal models for critical systems specification
  - automata
  - Petri nets
  - their extensions
- use model-checking to verify their properties
  - reachability
  - deadlocks
  - properties expressed in LTL and CTL logics
Outline

1 Automata
   - Introductory notions
     - Automata
     - Execution and execution tree
     - Atomic properties
   - Formal definitions
     - Automata
     - Behaviour
   - Extensions of automata
     - Automata with variables
     - Synchronised product of automata
     - Synchronisation by message passing

2 Temporal logic
   - Language
   - LTL
     - Formal syntax and semantics
     - Illustration
     - Examples of LTL formulae
   - CTL
     - Formal syntax and semantics
     - Illustration
     - Examples of CTL formulae

3 Model-checking
   - CTL model-checking
   - LTL model-checking

4 Symbolic model-checking
   - Computation of state sets
   - Binary Decision Diagrams
   - Automata representation

5 Reachability Properties
   - Reachability in temporal logic
   - Computation of the reachability
Outline

1 Automata
   - Introductory notions
     - Automata
     - Execution and execution tree
     - Atomic properties
   - Formal definitions
     - Automata
     - Behaviour
   - Extensions of automata
     - Automata with variables
     - Synchronised product of automata
     - Synchronisation by message passing
Intuitively, an automaton is a machine evolving from one state to another under the action of transitions.
Intuitively, an automaton is a machine evolving from one state to another under the action of transitions.

Example: Digital clock

- - - - 17:00
Intuitively, an **automaton** is a machine evolving from one **state** to another under the action of **transitions**.

**Example: Digital clock**

- - - - \[\rightarrow\] 17:00 \[\rightarrow\] 17:01
Intuitively, an automaton is a machine evolving from one state to another under the action of transitions.

Example: Digital clock

17:00 → 17:01 → 17:02
Intuitively, an automaton is a machine evolving from one state to another under the action of transitions.

Example: Digital clock
Example: Modulo 3 counter

- counts 0, 1, 2
- initial value 0
- allows operations increment and decrement
Example: Modulo 3 counter

- counts 0, 1, 2
- initial value 0
- allows operations increment and decrement
Example: Modulo 3 counter

- counts 0, 1, 2
- initial value 0
- allows operations increment and decrement
Example: Modulo 3 counter

- counts 0, 1, 2
- initial value 0
- allows operations increment and decrement
Example: Modulo 3 counter

- counts 0, 1, 2
- initial value 0
- allows operations increment and decrement
Example: Digicode

- 3 keys A, B, C
- code to open door ABA
- if the wrong key is pressed the whole operation has to start again
Example: Digicode

- 3 keys A, B, C
- code to open door ABA
- if the wrong key is pressed the whole operation has to start again
Example: Digicode

- 3 keys A, B, C
- code to open door ABA
- if the wrong key is pressed the whole operation has to start again
Example: Digicode

- 3 keys A, B, C
- code to open door ABA
- if the **wrong key** is pressed the whole operation has to **start again**
Example: Digicode

- 3 keys A, B, C
- code to open door ABA
- if the wrong key is pressed the whole operation has to start again

![Diagram of Digicode Automaton]

Remark: The numbers in the states are the number of correct keys that have already been pressed.
Automata

Example: Digicode

- 3 keys A, B, C
- code to open door ABA
- if the wrong key is pressed the whole operation has to start again

Laure Petrucci
Example: Digicode

- 3 keys A, B, C
- code to open door ABA
- if the **wrong key** is pressed the whole operation has to **start again**
Example: Digicode

- 3 keys A, B, C
- code to open door ABA
- if the wrong key is pressed the whole operation has to start again

Remark: The numbers in the states are the number of correct keys that have already been pressed.
Executions of a model

**Execution**

An execution is a sequence of states describing a possible evolution of the system.
Executions of a model

An execution is a sequence of states describing a possible evolution of the system.

- 0123
- 001201
- 001123
Executions of a model

Execution

An execution is a sequence of states describing a possible evolution of the system.

Questions

- Which executions lead to opening the door?
- Is there a possible infinite execution?

- 0123
- 001201
- 001123
Executions of a model

Execution

An execution is a sequence of states describing a possible evolution of the system.

Questions

- Which executions lead to opening the door?
- Is there a possible infinite execution?

- All those that end in state 3
- For example 00000000...
Execution tree

A tree to represent all possible executions

- **root**: initial state of the automaton
- **children** of a node: its **immediate successors** (states accessible from the node in one step)
Execution tree

A tree to represent all possible executions

- **root**: initial state of the automaton
- **children** of a node: its immediate successors (states accessible from the node in one step)

The digicode example

![Diagram of a digicode example with states 0, 1, 2, 3 and transitions A, B, C, B,C]
Automata

Introductory notions

Execution tree

A tree to represent all possible executions

- **root**: initial state of the automaton
- **children** of a node: its immediate successors (states accessible from the node in one step)

The digicode example
Exercise

Execution tree for the modulo 3 counter

Laure Petrucci
Exercise

Execution tree for the modulo 3 counter
Atomic properties

- **Atomic properties** are elementary properties known to be true or false
- some atomic properties can be associated with each state
- used to define more complex properties

Digicode atomic properties

- $P_A$: A has just been pressed
- $P_B$: B has just been pressed
- $P_C$: C has just been pressed

Associate properties with states

<table>
<thead>
<tr>
<th>State</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>$P_A$</td>
</tr>
<tr>
<td>2</td>
<td>$P_B$</td>
</tr>
<tr>
<td>3</td>
<td>$P_A$, $P_B$, $P_C$</td>
</tr>
</tbody>
</table>

Prove the correct code was entered when the door opens

The door is open only in state 3. Its only predecessor is 2 and transition $A$ is used from state 2 to state 3. So $A$ is the last key pressed.

The only predecessor of 2 is 1, and transition $B$ was used.

State 1 has two possible predecessors: 0 and 1, and both used $A$.

Therefore, the code entered ends with $ABA$. 
Atomic properties

- **Atomic properties** are elementary properties known to be true or false
- some atomic properties can be associated with each state
- used to define more complex properties

**Digicode atomic properties**

- $P_A$: A has just been pressed
- $P_B$: B has just been pressed
- $P_C$: C has just been pressed
Atomic properties

- **Atomic properties** are elementary properties known to be true or false.
- Some atomic properties can be associated with each state.
- Used to define more complex properties.

**Digicode atomic properties**

- $P_A$: A has just been pressed
- $P_B$: B has just been pressed
- $P_C$: C has just been pressed

**Associate properties with states**

The door is open only in state 3. Its only predecessor is 2 and transition A is used from state 2 to state 3. So A is the last key pressed. The only predecessor of 2 is 1, and transition B was used. State 1 has two possible predecessors: 0 and 1, and both used A. Therefore, the code entered ends with ABA.
Atomic properties

- **Atomic properties** are elementary properties known to be true or false
- some atomic properties can be associated with each state
- used to define more complex properties

**Digicode atomic properties**
- \( P_A \): A has just been pressed
- \( P_B \): B has just been pressed
- \( P_C \): C has just been pressed

**Associate properties with states**

The door is open only in state 3. Its only predecessor is 2 and transition A is used from state 2 to state 3. So A is the last key pressed.

The only predecessor of 2 is 1, and transition B was used.

State 1 has two possible predecessors: 0 and 1, and both used A.

Therefore, the code entered ends with ABA.
Atomic properties

- **Atomic properties** are elementary properties known to be true or false
- Some atomic properties can be associated with each state
- Used to define more complex properties

### Digicode atomic properties

- $P_A$: A has just been pressed
- $P_B$: B has just been pressed
- $P_C$: C has just been pressed

### Associate properties with states

Prove the correct code was entered when the door opens

The door is open only in state 3. Its only predecessor is 2 and transition A is used from state 2 to state 3. So A is the last key pressed.

The only predecessor of 2 is 1, and transition B was used.

State 1 has two possible predecessors: 0 and 1, and both used A.

Therefore, the code entered ends with ABA.
Atomic properties

- **Atomic properties** are elementary properties known to be true or false.
- Some atomic properties can be associated with each state.
- Used to define more complex properties.

**Digicode atomic properties**
- $P_A$: A has just been pressed
- $P_B$: B has just been pressed
- $P_C$: C has just been pressed

**Associate properties with states**

**Prove the correct code was entered when the door opens**

The door is open only in state 3. Its only predecessor is 2 and transition A is used from state 2 to state 3. So A is the last key pressed.

The only predecessor of 2 is 1, and transition B was used.

State 1 has two possible predecessors: 0 and 1, and both used A. Therefore, the code entered ends with ABA.
Formal definition of automata

Let $\text{Prop}$ be a set of atomic propositions. An automaton is a tuple $\mathcal{A} = \langle Q, E, T, q_0, l, F \rangle$ such that:

- $Q$ is a finite set of states
- $E$ is a finite set of transition labels
- $T \subseteq Q \times E \times Q$ is a set of transitions
- $q_0$ is the initial state
- $l : Q \rightarrow 2^{\text{Prop}}$ associates with each state a finite set of atomic propositions
- $F$ is a set of final states
Example

The digicode example

\[
Q = \{0, 1, 2, 3\}
\]
\[
E = \{A, B, C\}
\]
\[
T = \{(0, A, 1), (0, B, 0), (0, C, 0), (1, A, 1), (1, B, 2), (1, C, 0), (2, A, 3), (2, B, 0), (2, C, 0), (3, A, 4)\}
\]
\[
q_0 = 0
\]
\[
\text{Prop} = \{P_A, P_B, P_C\}
\]
\[
l(0) = \emptyset, l(1) = \{P_A\}, l(2) = \{P_B\}, l(3) = \{P_A\}
\]
\[
F = \{3\}
\]
Example

The digicode example

- $Q = \{0, 1, 2, 3\}$
Example

The digicode example

- \( Q = \{0, 1, 2, 3\} \)
- \( E = \{A, B, C\} \)
Example

The digicode example

- $Q = \{0, 1, 2, 3\}$
- $E = \{A, B, C\}$
- $T = \{(0, A, 1), (0, B, 0), (0, C, 0), (1, A, 1), (1, B, 2), (1, C, 0), (2, A, 3), (2, B, 0), (2, C, 0)\}$
Example

The digicode example

- $Q = \{0, 1, 2, 3\}$
- $E = \{A, B, C\}$
- $T = \{(0, A, 1), (0, B, 0), (0, C, 0), (1, A, 1), (1, B, 2), (1, C, 0), (2, A, 3), (2, B, 0), (2, C, 0)\}$
- $q_0 = 0$
Example

The digicode example

- $Q = \{0, 1, 2, 3\}$
- $E = \{A, B, C\}$
- $T = \{(0, A, 1), (0, B, 0), (0, C, 0), (1, A, 1), (1, B, 2), (1, C, 0), (2, A, 3), (2, B, 0), (2, C, 0)\}$
- $q_0 = 0$
- $Prop = \{P_A, P_B, P_C\}$
Example

The digicode example

- $Q = \{0, 1, 2, 3\}$
- $E = \{A, B, C\}$
- $T = \{(0, A, 1), (0, B, 0), (0, C, 0), (1, A, 1), (1, B, 2), (1, C, 0), (2, A, 3), (2, B, 0), (2, C, 0)\}$
- $q_0 = 0$
- $Prop = \{P_A, P_B, P_C\}$
- $l(0) = \emptyset$, $l(1) = \{P_A\}$, $l(2) = \{P_B\}$, $l(3) = \{P_A\}$
Example

The digicode example

- $Q = \{0, 1, 2, 3\}$
- $E = \{A, B, C\}$
- $T = \{(0, A, 1), (0, B, 0), (0, C, 0), (1, A, 1), (1, B, 2), (1, C, 0), (2, A, 3), (2, B, 0), (2, C, 0)\}$
- $q_0 = 0$
- $Prop = \{P_A, P_B, P_C\}$
- $l(0) = \emptyset$, $l(1) = \{P_A\}$, $l(2) = \{P_B\}$, $l(3) = \{P_A\}$
- $F = \{3\}$
Exercise

Formal representation of the modulo 3 counter (no property)
Exercise

Formal representation of the modulo 3 counter (no property)

- \( Q = \{0, 1, 2\} \)
- \( E = \{inc, dec\} \)
- \( T = \{(0, inc, 1), (0, dec, 2), (1, inc, 2), (1, dec, 0), (2, inc, 0), (2, dec, 1)\} \)
- \( q_0 = 0 \)
- \( Prop = \emptyset \)
- \( l(0) = l(1) = l(2) = \emptyset \)
- \( F = \emptyset \)
Behaviour

Runs (or paths)

- A run (or path) of an automaton $A$ is a sequence $\sigma$ of successive transitions $(q_i, e_i, q'_i)$ of $A$, i.e. such that $\forall i, q_{i+1} = q'_i$.

$$\sigma = q_1 \xrightarrow{e_1} q_2 \xrightarrow{e_2} q_3 \xrightarrow{e_3} q_4 \ldots$$

- The length of a run $\sigma$ is its number of transitions $|\sigma| \in \mathbb{N} \cup \{\omega\}$ where $\omega$ denotes infinity.

- The $i^{th}$ state of $\sigma$ is the state $q_{i+1}$ reached after $i$ transitions.
Behaviours

Runs (or paths)

- A run (or path) of an automaton $A$ is a sequence $\sigma$ of successive transitions $(q_i, e_i, q'_i)$ of $A$, i.e. such that $\forall i, q_{i+1} = q'_i$.

$$\sigma = q_1 \xrightarrow{e_1} q_2 \xrightarrow{e_2} q_3 \xrightarrow{e_3} q_4 \ldots$$

- The length of a run $\sigma$ is its number of transitions $|\sigma| \in \mathbb{N} \cup \{\omega\}$ where $\omega$ denotes infinity.

- The $i^{th}$ state of $\sigma$ is the state $q_{i+1}$ reached after $i$ transitions.

Executions

- A partial execution of $A$ is a run starting from the initial state $q_0$.

- A complete execution of $A$ is an execution that is maximal. It is either infinite or ends in a state where no transition is possible. This state might be final (in $F$), or a deadlock.

- A state is reachable if there exists an execution in which it appears.

- The complete executions define the behaviour of the automaton.
Exercise

Mutual exclusion between two processes

- two processes execute and need access to the same resource
- each process can request access to a critical section of its code
- they must not execute this part at the same time
- when they have finished they signal they exit their critical section and loop back to their initial state

Questions

1. Model this problem with an automaton
2. Associate atomic properties with each state
3. Is the mutual exclusion requirement satisfied?
4. Is the system fair?
5. What would happen if you wanted to add a third process?
Exercise

Automata

Formal definitions

Pᵢ: Process i is requesting access, Cᵢ: Process i is in its critical section, Rᵢ: Process i is at rest.

P₁: states 1, 3, 7; P₂: states 2, 3, 6; C₁: states 4, 6; C₂: states 5, 7; R₁: states 0, 2, 5; R₂: states 0, 1, 4

Yes: no state has both C₁ and C₂

No: run 0137137. . . never allows process 1 to enter its critical section

The number of states would blow up
Exercise

2. $P_i$: Process $i$ is requesting access,  
   $C_i$: Process $i$ is in its critical section,  
   $R_i$: Process $i$ is at rest.

$P_1$: states 1, 3, 7; $P_2$: states 2, 3, 6;  
$C_1$: states 4, 6; $C_2$: states 5, 7;  
$R_1$: states 0, 2, 5; $R_2$: states 0, 1, 4
Exercise

2. \( P_i \): Process \( i \) is requesting access,
   \( C_i \): Process \( i \) is in its critical section,
   \( R_i \): Process \( i \) is at rest.

   \( P_1 \): states 1, 3, 7; \( P_2 \): states 2, 3, 6;
   \( C_1 \): states 4, 6; \( C_2 \): states 5, 7;
   \( R_1 \): states 0, 2, 5; \( R_2 \): states 0, 1, 4

3. Yes: no state has both \( C_1 \) and \( C_2 \)
Exercise

\( P_i \): Process \( i \) is requesting access,
\( C_i \): Process \( i \) is in its critical section,
\( R_i \): Process \( i \) is at rest.

\( P_1 \): states 1, 3, 7; \( P_2 \): states 2, 3, 6;
\( C_1 \): states 4, 6; \( C_2 \): states 5, 7;
\( R_1 \): states 0, 2, 5; \( R_2 \): states 0, 1, 4

3 Yes: no state has both \( C_1 \) and \( C_2 \)

4 No: run 0137137... never allows process 1 to enter its critical section
Exercise

- $P_i$: Process $i$ is requesting access,
- $C_i$: Process $i$ is in its critical section,
- $R_i$: Process $i$ is at rest.

- $P_1$: states 1, 3, 7; $P_2$: states 2, 3, 6;
- $C_1$: states 4, 6; $C_2$: states 5, 7;
- $R_1$: states 0, 2, 5; $R_2$: states 0, 1, 4

- Yes: no state has both $C_1$ and $C_2$
- No: run 0137137... never allows process 1 to enter its critical section
- The number of states would blow up
Extension with variables

Why and how to use variables?
- more compact models, improving readability
- guards and assignments on transitions
Extension with variables

Why and how to use variables?
- more **compact** models, improving **readability**
- **guards** and **assignments** on transitions

Example: The digicode limited to 3 errors
Extension with variables

Why and how to use variables?

- more **compact** models, improving **readability**
- **guards** and **assignments** on transitions

Example: The digicode limited to 3 errors

```plaintext
var ctr: int;

ctr := 0
```
Extension with variables

Why and how to use variables?

- more compact models, improving readability
- guards and assignments on transitions

Example: The digicode limited to 3 errors

```plaintext
var ctr: int;

ctr := 0

B,C
ctr := ctr + 1

A
ctr := ctr + 1

B,C
ctr := ctr + 1

err
```

Diagram:

- States: 0, 1, 2, 3
- Transitions:
  - 0 -> 1 with input A
  - 0 -> 1 with input C
  - 1 -> 2 with input A
  - 2 -> 3 with input A
  - 2 -> 3 with input B
  - 3 back to 0 with input B,C
  - 3 back to 1 with input B,C
  - 3 back to 2 with input B,C
- Initial state: 0
- Final state: 3

```plaintext
ctr := 0

B,C
ctr := ctr + 1

A
ctr := ctr + 1

B,C
ctr := ctr + 1

err
```
Extension with variables

Why and how to use variables?

- more **compact** models, improving **readability**
- **guards** and **assignments** on transitions

Example: The digicode limited to 3 errors

```plaintext
var ctr: int;

ctr := 0

ctr < 3
B,C
ctr := ctr + 1

ctr < 3
C
ctr := ctr + 1

ctr < 3
B,C
ctr := ctr + 1

err

A

B

C

A

0

1

2

3
```
Extension with variables

Why and how to use variables?

- more compact models, improving readability
- guards and assignments on transitions

Example: The digicode limited to 3 errors

```plaintext
var ctr: int;

ctr := 0

ctr < 3, B, C
ctr := ctr + 1

ctr = 3
B, C
ctr := ctr + 1

ctr < 3, C
ctr := ctr + 1

ctr < 3, B, C
ctr := ctr + 1

err

0 1 2 3

A

ctr < 3
C
ctr := ctr + 1

ctr < 3
B, C
ctr := ctr + 1

A

A

B

A

Laure Petrucci

Advanced modelling techniques

18 / 95
Extension with variables

Why and how to use variables?

- more compact models, improving readability
- guards and assignments on transitions

Example: The digicode limited to 3 errors

```plaintext
var ctr: int;

ctr := 0

A

ctr < 3
B,C
ctr := ctr + 1

ctr < 3
B,C
ctr := ctr + 1

ctr = 3
A,C
ctr := ctr + 1;

ctr = 3
A,C
ctr := ctr + 1;

err

ctr < 3
B,C
ctr := ctr + 1

ctr < 3
B,C
ctr := ctr + 1

A

B

2

3

ctr < 3
C
ctr := ctr + 1

ctr < 3
B,C
ctr := ctr + 1
```
Exercise: The digicode with 3 errors without variables
Extension with variables

Exercise: The digicode with 3 errors without variables
Synchronised product

Why?

- each component of the system is designed as an automaton
- composition of automata
**Synchronised product**

**Why?**
- each component of the system is designed as an automaton
- composition of automata

**How?**
- **independent** actions lead to a cartesian product of states
- **synchronised** actions occur simultaneously
Synchronised product

3 counters, modulo 2, 3, 4: states

<table>
<thead>
<tr>
<th>State</th>
<th>0,0,0</th>
<th>0,1,0</th>
<th>1,0,0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,0,1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0,0,2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0,0,3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0,1,1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0,1,2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0,1,3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1,0,1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1,0,2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1,0,3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1,1,1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1,1,2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1,1,3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1,2,1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1,2,2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1,2,3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3 counters: some transitions

- inc,-,inc
- inc,inc,-
- inc,inc,inc
- inc,-inc
- inc,inc,-
- inc,-,-
- inc,inc,-
- inc,inc,inc
Example: Synchronised counters

Modulo 2 counter

- States: 0, 1
- Transitions: inc, dec

Modulo 3 counter

- States: 0, 1, 2
- Transitions: inc, dec

Modulo 4 counter

- States: 0, 1, 2, 3
- Transitions: inc, dec
Example: Synchronised counters

Modulo 2 counter

Modulo 3 counter

Modulo 4 counter

Synchronised actions: all counters increment or decrement simultaneously
Example: Synchronised counters

Modulo 2 counter

Modulo 3 counter

Modulo 4 counter

Synchronised actions: all counters increment or decrement simultaneously
## Formal definition of the cartesian product

Let \((A_i)_{1 \leq i \leq n}\) be a family of automata \(A_i = \langle Q_i, E_i, T_i, q_{0i}, l_i, F_i \rangle\).

### Cartesian product of automata

The **cartesian product** \(A_1 \times \cdots \times A_n\) of the automata in the family is the automaton \(A = \langle Q, E, T, q_0, l, F \rangle\) such that:

- \(Q = Q_1 \times \cdots \times Q_n\)
- \(E = \prod_{1 \leq i \leq n}(E_i \cup \{-\})\) (where \(-\) represents an **empty action**)
- \(T = \{(((q_1, \ldots, q_n), (e_1, \ldots, e_n), (q'_1, \ldots, q'_n)) |\)
  \(\forall 1 \leq i \leq n, (e_i = - \land q'_i = q_i) \lor (e_i \neq - \land (q_i, e_i, q'_i) \in T_i)\}\)
- \(q_0 = (q_{01}, \ldots, q_{0n})\)
- \(\forall (q_1, \ldots, q_n) \in Q : l(((q_1, \ldots, q_n))) = \bigcup_{1 \leq i \leq n} l_i(q_i)\)
- \(F = \{(q_1, \ldots, q_n) \in Q | \exists 1 \leq i \leq n, q_i \in F_i\}\)
Formal definition of the synchronised product

Let \((\mathcal{A}_i)_{1 \leq i \leq n}\) be a family of automata \(\mathcal{A}_i = \langle Q_i, E_i, T_i, q_{0i}, l_i, F_i \rangle\).

**Synchronisation set**

The **synchronisation set**, denoted \(Sync\) describes all permitted simultaneous actions:

\[
Sync \subseteq \prod_{1 \leq i \leq n} (E_i \cup \{-\})
\]
Formal definition of the synchronised product

Let \((A_i)_{1 \leq i \leq n}\) be a family of automata \(A_i = \langle Q_i, E_i, T_i, q_{0i}, l_i, F_i \rangle\).

**Synchronisation set**

The synchronisation set, denoted \(Sync\) describes all permitted simultaneous actions:

\[
Sync \subseteq \prod_{1 \leq i \leq n} (E_i \cup \{-\})
\]

**Synchronised product of automata**

The synchronised product of \((A_i)_{1 \leq i \leq n}\) over a set \(Sync\) is the cartesian product restricted to \(E = Sync\).
Synchronisation by message passing

Message passing: a special case of synchronised product

!m send a message m

?m receive a message m

- reception and sending occur simultaneously
- they concern the same message
Synchronisation by message passing

Example: a small lift

Model of a lift in a 3 floors building, composed of:

- **the cabin** which goes up and down according to the current floor and the lift controller commands
- **3 doors** (one per floor) which open and close according to the controller’s commands
- **a controller** which operates the lift
Example: the lift

**Cabin**

- Ith door: $C_i$ to $O_i$
- $i$th door: ?close$_i$, ?open$_i$
- Cabin: ?down, ?up, ?up
- Floor transition:
  - Floor 0: 0 → 1
  - Floor 1: 1 → 2
  - Floor 2: 2 → 0

**Controller**

- Floor transition:
  - Floor 0: 0 → 2
  - Floor 1: 1 → 2
  - Floor 2: 2 → 0

- Events:
  - open$_0$, close$_0$
  - open$_1$, close$_1$
  - open$_2$, close$_2$
  - up, down
Example: the lift

Cabin

\[ \begin{array}{c}
\text{0} \\
\text{up} \\
\text{down} \\
\text{1} \\
\text{up} \\
\text{down} \\
\text{2} \\
\text{up} \\
\text{down}
\end{array} \]

Controller

\[ \begin{array}{c}
\text{0} \\
\text{2} \\
\text{at0} \\
\text{at1} \\
\text{at2} \\
\text{floor0} \\
\text{floor1} \\
\text{floor2}
\end{array} \]

\[ \begin{array}{c}
\text{!up} \\
\text{!down} \\
\text{!close} \\
\text{!open}
\end{array} \]

Properties

- A door on a floor cannot open while the cabin is on a different floor
- The cabin cannot move while one of the doors is open
Exercise

Mutual exclusion problem

1. Model the mutual exclusion problem with message passing:
   - one automaton per participating process (2 processes)
   - a controller

2. How do you add a new process? Give the model for 3 processes, and explain how to generalise it to $n$ processes
Exercise

Mutual exclusion problem

1. Model the mutual exclusion problem with message passing:
   - one automaton per participating process (2 processes)
   - a controller

2. How do you add a new process? Give the model for 3 processes, and explain how to generalise it to \( n \) processes

process \( i, \ i \in \{1, 2\} \)

- \( \text{idle}_i \) → \( \text{req}_i \) → \( \text{enter}_i \) → \( \text{cs}_i \) → \( \text{end}_i \)

controller

- \( \text{free} \) → \( \text{occ}_1 \) → \( \text{occ}_2 \) → \( \text{request}_1 \) → \( \text{request}_2 \) → \( \text{end}_1 \) → \( \text{end}_2 \)
Exercise

Mutual exclusion problem

1. Model the mutual exclusion problem with message passing:
   - one automaton per participating process (2 processes)
   - a controller

2. How do you add a new process? Give the model for 3 processes, and explain how to generalise it to $n$ processes
Exercise

Mutual exclusion problem

1. Model the mutual exclusion problem with message passing:
   - one automaton per participating process (2 processes)
   - a controller

2. How do you add a new process? Give the model for 3 processes, and explain how to generalise it to \( n \) processes

**process \( i, i \in \{1, 2, 3\} \)**

- idle\(_i\) \( \rightarrow \) req\(_i\) \( \rightarrow \) enter\(_i\) \( \rightarrow \) cs\(_i\)
- \( !request_i \)
- \( !end_i \)

**controller**

- \( n \) process automata
- controller: \( n \) states occ
Outline

Temporal logic

- Language
- LTL
  - Formal syntax and semantics
  - Illustration
  - Examples of LTL formulae
- CTL
  - Formal syntax and semantics
  - Illustration
  - Examples of CTL formulae
Introduction to temporal logics

- express **dynamic behaviour** of the system
- use **formal syntax and semantics** to avoid any ambiguity
- capture statements and reasoning that involve the notion of **order in time**

The lift example

- any request must ultimately be satisfied
- False: The lift can continuously go up and down without opening doors (run $\text{up} \rightarrow \text{up} \rightarrow \text{down} \rightarrow \text{up}$)
- False: consequence of the previous property
Introduction to temporal logics

- express dynamic behaviour of the system
- use formal syntax and semantics to avoid any ambiguity
- capture statements and reasoning that involve the notion of order in time

The lift example

- any request must ultimately be satisfied

- the lift never traverses a floor for which a request is pending without satisfying the request
Introduction to temporal logics

- express **dynamic behaviour** of the system
- use **formal syntax and semantics** to avoid any ambiguity
- capture statements and reasoning that involve the notion of **order in time**

The lift example

- any request must ultimately be satisfied
  False: The lift can continuously go up and down without opening doors
  \[
  (\text{run } (at0,C_0,C_1,C_2,0) \xrightarrow{up} (at1,C_0,C_1,C_2,1) \xrightarrow{up} (at2,C_0,C_1,C_2,2) \xrightarrow{down} (at1,C_0,C_1,C_2,1) \ldots)
  \]
- the lift never traverses a floor for which a request is pending without satisfying the request
Introduction to temporal logics

- express **dynamic behaviour** of the system
- use **formal syntax and semantics** to avoid any ambiguity
- capture statements and reasoning that involve the notion of **order in time**

**The lift example**

- any request must ultimately be satisfied
  False: The lift can continuously go up and down without opening doors
  \[
  \text{run} \left( \text{at}0, C_0, C_1, C_2, 0 \right) \xrightarrow{up} \left( \text{at}1, C_0, C_1, C_2, 1 \right) \xrightarrow{up} \left( \text{at}2, C_0, C_1, C_2, 2 \right) \xrightarrow{down} \left( \text{at}1, C_0, C_1, C_2, 1 \right) \ldots \]
- the lift never traverses a floor for which a request is pending without satisfying the request
  False: consequence of the previous property
The language CTL*

- **atomic propositions**
- **boolean combinators:**
  - true, false
  - \( \neg \) (negation)
  - \( \land \) (and), \( \lor \) (or)
  - \( \implies \) (logical implication), \( \iff \) (if and only if)
- **temporal combinators:**
  - X (neXt), F (Future), G (Globally)
  - U (Until), W (Weak until)
- **quantifiers:** A (Always), E (Exists)
The language CTL*

- **atomic propositions**
- **boolean combinators:**
  - `true`, `false`
  - `¬` (negation)
  - `∧` (and), `∨` (or)
  - `⇒` (logical implication), `⇔` (if and only if)
- **temporal combinators:**
  - `X` (neXt), `F` (Future), `G` (Globally)
  - `U` (Until), `W` (Weak until)
- **quantifiers:** `A` (Always), `E` (Exists)

**Main temporal logics**

- **LTL** Linear-time Temporal Logic
- **CTL** Computation Tree Logic
LTL: Linear-time Temporal Logic
Semantics of LTL

Let $\sigma$ be a run and $p \in Prop$ an atomic proposition. $\sigma, i \models \phi$ denotes that at time $i$ of its execution, $\sigma$ satisfies formula $\phi$.

<table>
<thead>
<tr>
<th>$\sigma, i \models p$</th>
<th>iff</th>
<th>$p \in l(\sigma(i))$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma, i \models \neg \phi$</td>
<td>iff</td>
<td>$\sigma, i \not\models \phi$</td>
</tr>
<tr>
<td>$\sigma, i \models \phi \land \psi$</td>
<td>iff</td>
<td>$\sigma, i \models \phi$ and $\sigma, i \models \psi$</td>
</tr>
<tr>
<td>$\sigma, i \models X\phi$</td>
<td>iff</td>
<td>$i &lt;</td>
</tr>
<tr>
<td>$\sigma, i \models F\phi$</td>
<td>iff</td>
<td>$\exists j, i \leq j \leq</td>
</tr>
<tr>
<td>$\sigma, i \models G\phi$</td>
<td>iff</td>
<td>$\forall j, i \leq j \leq</td>
</tr>
<tr>
<td>$\sigma, i \models \phi U \psi$</td>
<td>iff</td>
<td>$\exists j, i \leq j \leq</td>
</tr>
</tbody>
</table>
Illustration of the LTL semantics

- $p$
- $X\phi$
- $F\phi$
- $G\phi$
- $\phi_1 U \phi_2$
Examples of LTL formulae

What do the following formulae mean?
Which runs satisfy the LTL property?

Modulo 3 counter

1. XXX0

2. F(1 ∨ 2)

3. F1
Examples of LTL formulae

- What do the following formulae mean?
- Which runs satisfy the LTL property?

Modulo 3 counter

1. XXX0
   The third state reached is 0

2. F(1 ∨ 2)

3. F1
Examples of LTL formulae

- What do the following formulae mean?
- Which runs satisfy the LTL property?

### Modulo 3 counter

1. **XXX0**
   - The third state reached is 0
   - All runs starting with 0120 or 0210

2. **F(1 ∨ 2)**

3. **F1**
Examples of LTL formulae

What do the following formulae mean?
Which runs satisfy the LTL property?

Modulo 3 counter

1. XXX0
   The third state reached is 0
   All runs starting with 0120 or 0210

2. F(1 ∨ 2)
   In the future state 1 or state 2 will be reached

3. F1
Examples of LTL formulae

- What do the following formulae mean?
- Which runs satisfy the LTL property?

**Modulo 3 counter**

1. **XXX0**
   - The third state reached is 0
   - All runs starting with 0120 or 0210

2. **F(1 ∨ 2)**
   - In the future state 1 or state 2 will be reached
   - All runs

3. **F1**
Examples of LTL formulae

- What do the following formulae mean?
- Which runs satisfy the LTL property?

### Modulo 3 counter

**1** XXX0
- The third state reached is 0
- All runs starting with 0120 or 0210

**2** F(1 ∨ 2)
- In the future state 1 or state 2 will be reached
- All runs

**3** F1
- In the future state 1 will be reached
Examples of LTL formulae

- What do the following formulae mean?
- Which runs satisfy the LTL property?

## Modulo 3 counter

1. **XXX0**
   - The third state reached is 0
   - All runs starting with 0120 or 0210

2. **F(1 ∨ 2)**
   - In the future state 1 or state 2 will be reached
   - All runs

3. **F1**
   - In the future state 1 will be reached
   - All runs containing 1, i.e. all runs except 020202...
Examples of LTL formulae

- What do the following formulae mean?
- Which runs satisfy the LTL property?

The digicode

1. $F3$
2. $G\neg3$
Examples of LTL formulae

- What do the following formulae mean?
- Which runs satisfy the LTL property?

The digicode

1. F3
   - The door can open

2. G¬3
Examples of LTL formulae

- What do the following formulae mean?
- Which runs satisfy the LTL property?

The digicode

1. F3
   - The door can open
   - All runs ending in state 3

2. G¬3
Examples of LTL formulae

- What do the following formulae mean?
- Which runs satisfy the LTL property?

The digicode

1. F3
   The door can open
   All runs ending in state 3

2. G¬3
   The door never opens
Examples of LTL formulae

- What do the following formulae mean?
- Which runs satisfy the LTL property?

The digicode

1. F3
   - The door can open
   - All runs ending in state 3

2. G¬3
   - The door never opens
   - All runs not ending in state 3
Exercises

Express $\lor$, $\equiv$, $\iff$, $W$ with $\neg$, $\land$, $X$, $F$, $G$, $U$

($W$ is similar to $U$ but $\psi$ may never happen)
Exercises

Express $\lor$, $\implies$, $\iff$, $W$ with $\neg$, $\land$, $X$, $F$, $G$, $U$

($W$ is similar to $U$ but $\psi$ may never happen)

- $\phi \lor \psi \equiv \neg (\phi \land \psi)$
- $\phi \implies \psi \equiv \neg \phi \lor \psi$
- $\phi \iff \psi \equiv (\neg \phi \lor \psi) \land (\phi \lor \neg \psi)$
- $\phi W \psi \equiv (\phi U \psi) \lor G \phi$
Prove that:

1. $F\phi \equiv \text{true} \cup \phi$

2. $G\phi \equiv \neg F \neg \phi$
Exercises

Prove that:

1. $F\phi \equiv \text{true}\text{U}\phi$
   
   $\text{true}\text{U}\phi \equiv \exists j, i \leq j \leq |\sigma| : \sigma, j \models \phi \land \forall k, i \leq k < j : \sigma, k \models \text{true}$
   
   $\equiv \exists j, i \leq j \leq |\sigma| : \sigma, j \models \phi$
   
   $\equiv F\phi$

2. $G\phi \equiv \neg F\neg\phi$
Exercises

Prove that:

1. \( F\phi \equiv \text{trueU}\phi \)
   \[
   \text{trueU}\phi \equiv \exists j, i \leq j \leq |\sigma| : \sigma, j \models \phi \land \forall k, i \leq k < j : \sigma, k \models \text{true}
   \]
   \[
   \equiv \exists j, i \leq j \leq |\sigma| : \sigma, j \models \phi
   \]
   \[
   \equiv F\phi
   \]

2. \( G\phi \equiv \neg F\neg\phi \)
   \[
   \neg F\neg\phi \equiv \neg(\exists j, i \leq j \leq |\sigma| : \sigma, j \models \neg\phi)
   \]
   \[
   \equiv \forall j, i \leq j \leq |\sigma| : \sigma, j \not\models \neg\phi
   \]
   \[
   \equiv \forall j, i \leq j \leq |\sigma| : \sigma, j \models \phi
   \]
   \[
   \equiv G\phi
   \]
## Exercises

### Digicode

1. Write a LTL formula satisfied by all runs where keys A and B have successively been pressed

2. Write a LTL formula that characterises the infinite loop on state 0

3. Same question using atomic propositions $P_A$, $P_B$, $P_C$
Exercises

Digicode

1. Write a LTL formula satisfied by all runs where keys A and B have successively been pressed:
   \[ F(P_A \implies XP_B) \]

2. Write a LTL formula that characterises the infinite loop on state 0

3. Same question using atomic propositions \( P_A, P_B, P_C \)
Exercises

Digicode

1. Write a LTL formula satisfied by all runs where keys A and B have successively been pressed:
   $$F(P_A \implies XP_B)$$

2. Write a LTL formula that characterises the infinite loop on state 0:
   $$G_0$$

3. Same question using atomic propositions $P_A$, $P_B$, $P_C$
Exercises

Digicode

1. Write a LTL formula satisfied by all runs where keys A and B have successively been pressed
   \[ F(P_A \implies XP_B) \]

2. Write a LTL formula that characterises the infinite loop on state 0
   \[ G0 \]

3. Same question using atomic propositions \( P_A, P_B, P_C \)
   \[ G\neg P_A \]
Mutual exclusion between two processes (synchronised product)

Write an LTL formula satisfied by all runs where:

1. The two processes are not simultaneously in their critical section

2. Whenever process 1 requests to enter its critical section, it will eventually succeed
Mutual exclusion between two processes (synchronised product)

Write an LTL formula satisfied by all runs where:

1. The two processes are not simultaneously in their critical section
   \[ G\neg (cs_1 \land cs_2) \]
2. Whenever process 1 requests to enter its critical section, it will eventually succeed
Exercises

Mutual exclusion between two processes (synchronised product)

Write an LTL formula satisfied by all runs where:

1. The two processes are not simultaneously in their critical section
   \[ \neg G(cs_1 \land cs_2) \]

2. Whenever process 1 requests to enter its critical section, it will eventually succeed
   \[ G(req_1 \implies Fcs_1) \]
CTL: Computation Tree Logic
Semantics of CTL

**Same as LTL plus:**

\[
\begin{align*}
\sigma, i \models \text{E}\phi & \iff \exists \sigma' : \sigma(0) \ldots \sigma(i) = \sigma'(0) \ldots \sigma'(i) \text{ and } \sigma', i \models \phi \\
\sigma, i \models \text{A}\phi & \iff \forall \sigma' : \sigma(0) \ldots \sigma(i) = \sigma'(0) \ldots \sigma'(i) \text{ we have } \sigma', i \models \phi
\end{align*}
\]

In CTL, each use of a temporal operator (X, F, G, U) is in the immediate scope of a quantifier (E, A)

This restriction does not apply in CTL*
Illustration of the CTL semantics (1/8)
Illustration of the CTL semantics (2/8)
Illustration of the CTL semantics (3/8)

\[ E\phi U\psi \]
Illustration of the CTL semantics (4/8)
Illustration of the CTL semantics (5/8)

$$\text{AX} \phi$$
Illustration of the CTL semantics (6/8)
Illustration of the CTL semantics (7/8)
Illustration of the CTL semantics (8/8)
Examples of CTL formulae

Explain the following CTL formulae, and if they are true or false:

Mutual exclusion between 2 processes (synchronised product)

1. $\text{AG}\neg(cs_1 \land cs_2)$

Whatever happens, the two processes cannot be simultaneously in their critical section. **true**

2. $\text{AG}(\text{req}_1 \implies \text{AF}cs_1)$

It is always the case that when process 1 requests access to its critical section, it will eventually be granted. **false**

3. $\text{AG}(\text{EF}(\text{idle}_1 \land \text{idle}_2))$

Whatever the state of the system, it is possible to have both processes idle in the future. **true**
Examples of CTL formulae

Explain the following CTL formulae, and if they are true or false:

**Mutual exclusion between 2 processes (synchronised product)**

1. \( \text{AG} \neg (cs_1 \land cs_2) \)
   Whatever happens, the two processes cannot be simultaneously in their critical section
   true

2. \( \text{AG}(\text{req}_1 \implies \text{AF}cs_1) \)

3. \( \text{AG}(\text{EF}(idle_1 \land idle_2)) \)
Examples of CTL formulae

Explain the following CTL formulae, and if they are true or false:

**Mutual exclusion between 2 processes (synchronised product)**

1. $\text{AG}\neg(cs_1 \land cs_2)$
   
   Whatever happens, the two processes cannot be simultaneously in their critical section
   
   true

2. $\text{AG}(req_1 \implies AF cs_1)$
   
   It is always the case that when process 1 requests access to its critical section, it will eventually be granted
   
   false

3. $\text{AG}(EF(idle_1 \land idle_2))$
   
   Whatever the state of the system, it is possible to have both processes idle in the future.
   
   true
**Examples of CTL formulae**

Explain the following CTL formulae, and if they are true or false:

---

**Mutual exclusion between 2 processes (synchronised product)**

1. \(\text{AG}\neg(cs_1 \land cs_2)\)
   - Whatever happens, the two processes cannot be simultaneously in their critical section
   - **true**

2. \(\text{AG}(req_1 \implies \text{AF}cs_1)\)
   - It is always the case that when process 1 requests access to its critical section, it will eventually be granted
   - **false**

3. \(\text{AG}(\text{EF}(idle_1 \land idle_2))\)
   - Whatever the state of the system, it is possible to have both processes idle in the future.
   - **true**
---
Exercises

Prove that:

1. $\text{EF} \phi \equiv \text{E} \text{true} \text{U} \phi$

2. $\text{AX} \phi \equiv \neg \text{EX} \neg \phi$

3. $\text{AG} \phi \equiv \neg (\text{E} \text{true} \text{U} \neg \phi)$

4. $\text{AF} \phi \equiv \neg \text{EG} \neg \phi$
Exercises

Prove that:

1. \( EF\phi \equiv E\text{true}U\phi \)

   We already proved that \( F\phi \equiv \text{true}U\phi \). Hence: \( EF\phi \equiv E(\text{true}U\phi) \)

2. \( AX\phi \equiv \neg EX\neg\phi \)

3. \( AG\phi \equiv \neg (E\text{true}U\neg\phi) \)

4. \( AF\phi \equiv \neg EG\neg\phi \)
Exercises

Prove that:

1. \( EF\phi \equiv E(trueU\phi) \)
   
   We already proved that \( F\phi \equiv trueU\phi \). Hence: \( EF\phi \equiv E(trueU\phi) \)

2. \( AX\phi \equiv \neg EX\neg \phi \)
   
   \( \neg EX\neg \phi \equiv \neg(\exists \sigma' : \sigma(0) \ldots \sigma(i) = \sigma'(0) \ldots \sigma'(i) \land \sigma', i \models X\neg \phi) \)
   
   \( \equiv \forall \sigma' : \sigma(0) \ldots \sigma(i) = \sigma'(0) \ldots \sigma'(i) \text{ we have } \sigma', i \not\models X\neg \phi \)
   
   \( \equiv \forall \sigma' : \sigma(0) \ldots \sigma(i) = \sigma'(0) \ldots \sigma'(i) \text{ we have } \sigma', i \models X\phi \)
   
   \( \equiv AX\phi \)

3. \( AG\phi \equiv \neg (E(trueU\neg \phi)) \)

4. \( AF\phi \equiv \neg EG\neg \phi \)
Exercises

Prove that:

1. \( EF\phi \equiv E\text{true}U\phi \)
   
   We already proved that \( F\phi \equiv \text{true}U\phi \). Hence: \( EF\phi \equiv E(\text{true}U\phi) \)

2. \( AX\phi \equiv \neg EX\neg\phi \)
   
   \( \neg EX\neg\phi \equiv \neg(\exists \sigma' : \sigma(0) \ldots \sigma(i) = \sigma'(0) \ldots \sigma'(i) \land \sigma', i \models X\neg\phi) \)
   
   \( \equiv \forall \sigma' : \sigma(0) \ldots \sigma(i) = \sigma'(0) \ldots \sigma'(i) \) we have \( \sigma', i \not\models X\neg\phi \)
   
   \( \equiv \forall \sigma' : \sigma(0) \ldots \sigma(i) = \sigma'(0) \ldots \sigma'(i) \) we have \( \sigma', i \models X\phi \)
   
   \( \equiv AX\phi \)

3. \( AG\phi \equiv \neg(E\text{true}U\neg\phi) \)
   
   We know that \( EF\phi \equiv E\text{true}U\phi \) and \( G\phi \equiv \neg F\neg\phi \). Hence:
   
   \( \neg(E\text{true}U\neg\phi) \equiv \neg EF\neg\phi \)
   
   \( \equiv A\neg F\neg\phi \)
   
   \( \equiv AG\phi \)

4. \( AF\phi \equiv \neg EG\neg\phi \)
Exercises

Prove that:

1. $\text{EF}\phi \equiv \text{EtrueU}\phi$
   
   We already proved that $\text{F}\phi \equiv \text{trueU}\phi$. Hence: $\text{EF}\phi \equiv \text{E(trueU}\phi)$

2. $\text{AX}\phi \equiv \neg\text{EX}\neg\phi$
   
   $\neg\text{EX}\neg\phi \equiv \neg(\exists\sigma' : \sigma(0)\ldots\sigma(i) = \sigma'(0)\ldots\sigma'(i) \land \sigma', i \models \text{X}\neg\phi)$
   
   $\equiv \forall\sigma' : \sigma(0)\ldots\sigma(i) = \sigma'(0)\ldots\sigma'(i) \text{ we have } \sigma', i \not\models \text{X}\neg\phi$
   
   $\equiv \forall\sigma' : \sigma(0)\ldots\sigma(i) = \sigma'(0)\ldots\sigma'(i) \text{ we have } \sigma', i \models \text{X}\phi$
   
   $\equiv \text{AX}\phi$

3. $\text{AG}\phi \equiv \neg(\text{EtrueU}\neg\phi)$
   
   We know that $\text{EF}\phi \equiv \text{EtrueU}\phi$ and $\text{G}\phi \equiv \neg\text{F}\neg\phi$. Hence:
   
   $\neg(\text{EtrueU}\neg\phi) \equiv \neg\text{EF}\neg\phi$
   
   $\equiv \text{A}\neg\text{F}\neg\phi$
   
   $\equiv \text{AG}\phi$

4. $\text{AF}\phi \equiv \neg\text{EG}\neg\phi$
   
   $\neg\text{EG}\neg\phi \equiv \text{A}\neg\text{G}\neg\phi$
   
   $\equiv \text{AF}\phi$
LTL and CTL do not recognise the same behaviours

\[ \forall \phi : \mathcal{A}_1 \models \phi \iff \mathcal{A}_2 \not\models \phi \]

**LTL**

Runs for both automata:
1. \{P, Q\} \{P\} \{-\}
2. \{P, Q\} \{P\} \{Q\}

**CTL**

\[ \mathcal{A}_1 \models AX(\exists XQ \land \exists X\neg Q) \]
\[ \mathcal{A}_2 \not\models AX(\exists XQ \land \exists X\neg Q) \]
Outline

3 Model-checking
   - CTL model-checking
   - LTL model-checking
CTL model-checking algorithm

- algorithm **marking** states where a formula is satisfied
- **memorises** the already computed results
- **reuses** the computed results of sub-formulae to compute new formulae
**Procedure** marking(\(\phi\))

```plaintext
case \(\phi = p\) do
    forall \(q \in Q\) do
        if \(p \in l(q)\) then
            q.\(\phi\) := true
        else
            q.\(\phi\) := false
```

\(\phi = req_1\)
Procedure `marking(\(\phi\))`

```latex
\begin{align*}
\text{case } \phi = p \text{ do} \\
\quad \text{forall } q \in Q \text{ do} \\
\quad \quad \text{if } p \in l(q) \text{ then} \\
\quad \quad \quad q.\phi := \text{true} \\
\quad \quad \text{else} \\
\quad \quad \quad q.\phi := \text{false}
\end{align*}
```
Case 2: $\phi = \neg \psi$

marking($\psi$);
forall $q \in Q$ do
  $\underline{q.\phi:=\neg q.\psi}$

\[
\phi = \neg req_1
\]
Case 2: $\phi = \neg \psi$

marking($\psi$);
forall $q \in Q$ do
\[ q.\phi := \neg q.\psi \]
Case 2: $\phi = \neg \psi$

marking($\psi$);
for all $q \in Q$ do
    $\mid q.\phi := \neg q.\psi$

$\phi = \neg req_1$
Case 3: $\phi = \psi_1 \land \psi_2$

```
marking(\psi_1);
marking(\psi_2);
\textbf{forall} q \in Q \textbf{ do}
\qquad \text{\_} q.\phi := q.\psi_1 \land q.\psi_2
```
Case 3: $\phi = \psi_1 \land \psi_2$

marking($\psi_1$);  
marking($\psi_2$);  
forall $q \in Q$ do  
  $\downarrow q.\phi := q.\psi_1 \land q.\psi_2$

$\phi = req_1 \land req_2$
Case 3: $\phi = \psi_1 \land \psi_2$

marking($\psi_1$);
marking($\psi_2$);
forall $q \in Q$ do
  □ q.$\phi$ := q.$\psi_1$ $\land$ q.$\psi_2$
Case 3: $\phi = \psi_1 \land \psi_2$

marking($\psi_1$);
marking($\psi_2$);
forall $q \in Q$ do
  $\models q.\phi := q.\psi_1 \land q.\psi_2$

$\phi = req_1 \land req_2$
Case 4: $\phi = \text{EX}\psi$

marking($\psi$);

forall $q \in Q$ do

| q.$\phi$ := false

forall $(q, -, q') \in T$ do

| if $q'.\psi = \text{true}$ then

| q.$\phi$ := true

$\phi = \text{EX}\text{req}_1$
Case 4: $\phi = \text{EX}\psi$

marking($\psi$);

forall $q \in Q$ do
  $\underline{q.\phi := \text{false}}$

forall $(q,\_ , q') \in T$ do
  if $q'.\psi = \text{true}$ then
    $\underline{q.\phi := \text{true}}$

$\phi = \text{EXreq}_1$
Case 4: $\phi = \text{EX}\psi$

marking($\psi$);
\[
\text{forall } q \in Q \text{ do}
\begin{align*}
\quad & q.\phi := \text{false} \\
\end{align*}
\]
\[
\text{forall } (q, -, q') \in T \text{ do}
\begin{align*}
\quad & \text{if } q'.\psi = \text{true then} \\
\quad & q.\phi := \text{true}
\end{align*}
\]
Case 4: $\phi = \text{EX} \psi$

marking($\psi$);
forall $q \in Q$ do
  $\underline{\_} \ q.\phi := \text{false}$
forall $(q, \_ , q') \in T$ do
  if $q'.\psi = \text{true}$ then
  $\underline{\_} \ q.\phi := \text{true}$
  $\underline{\_}$

\[\phi = \text{EX} req_1\]
Case 5: $\phi = E\psi_1 U \psi_2$

marking($\psi_1$);
marking($\psi_2$);
forall $q \in Q$ do
  q.$\phi$:=false;
  q.seenbefore:=false
L:=\emptyset;
forall $q \in Q$ do
  if q.$\psi_2$=true then
    L:=L∪{q}
while L≠\emptyset do
  pick q from L; L:=L\{q};
  q.$\phi$:=true;
forall (q', -, q) ∈ T do
  if q'.seenbefore=false then
    q'.seenbefore:=true;
    if q'.$\psi_1$=true then
      L:=L∪{q'}
Case 5: $\phi = E\psi_1 U\psi_2$

marking($\psi_1$);
marking($\psi_2$);
forall $q \in Q$ do
  q.$\phi$:=false;
  q.seenbefore:=false
L:=∅;
forall $q \in Q$ do
  if q.$\psi_2$=true then
    L:=L∪{q}
while L≠∅ do
  pick q from L; L:=L\{q};
  q.$\phi$:=true;
  forall (q′, →, q) ∈ $T$ do
    if q′.seenbefore=false then
      q′.seenbefore:=true;
      if q′.$\psi_1$=true then
        L:=L∪{q′}
Case 5: $\phi = E\psi_1 U \psi_2$

marking($\psi_1$);
marking($\psi_2$);
forall $q \in Q$ do
  $q.\phi := false$;
  $q.\text{seenbefore} := false$
L := $\emptyset$;
forall $q \in Q$ do
  if $q.\psi_2 = true$ then
    L := L \cup \{q\}$
while L \neq \emptyset do
  pick q from L; L := L \setminus \{q\};
  $q.\phi := true$;
  forall ($q', \rightarrow, q$) \in T do
    if $q'.\text{seenbefore} = false$ then
      $q'.\text{seenbefore} := true$;
      if $q'.\psi_1 = true$ then
        L := L \cup \{q'\}$

\[
\phi = E\text{req}_1 U \text{cs}_1
\]
Case 5: $\phi = E\psi_1 U \psi_2$

marking($\psi_1$);
marking($\psi_2$);
for all $q \in Q$ do
  $q.\phi := false$;
  $q.seenbefore := false$
L := $\emptyset$;
for all $q \in Q$ do
  if $q.\psi_2 = true$ then
    L := L $\cup$ \{q\}
while L $\neq \emptyset$ do
  pick q from L; L := L \{q\};
  q.\phi := true;
  for all $(q',\_ ,q) \in T$ do
    if q'.seenbefore = false then
      q'.seenbefore := true;
      if q'.\psi_1 = true then
        L := L $\cup$ \{q'\}

$\phi = Ereq_1 U cs_1$

<table>
<thead>
<tr>
<th>seenbefore</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td></td>
</tr>
</tbody>
</table>
Case 5: $\phi = E\psi_1 U \psi_2$

marking($\psi_1$);
marking($\psi_2$);
foreach $q \in Q$ do
  $q.\phi := false$;
  $q.\text{seenbefore} := false$
L := $\emptyset$;
foreach $q \in Q$ do
  if $q.\psi_2 = true$ then
    L := L $\cup \{q\}$
while $L \neq \emptyset$ do
  pick q from L; L := L $\setminus \{q\}$;
  $q.\phi := true$;
  foreach $(q', \ldots, q) \in T$ do
    if $q'.\text{seenbefore} = false$ then
      $q'.\text{seenbefore} := true$;
      if $q'.\psi_1 = true$ then
        L := L $\cup \{q'\}$

<table>
<thead>
<tr>
<th>seenbefore</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>${4,6}$</td>
</tr>
</tbody>
</table>
Case 5: $\phi = E \psi_1 U \psi_2$

marking($\psi_1$);
marking($\psi_2$);
forall $q \in Q$ do
  q.$\phi$ := false;
  q.seenbefore := false
L := {};
forall $q \in Q$ do
  if $q.\psi_2$ = true then
    L := L \cup {q}
while L \neq {} do
  pick q from L; L := L \setminus {q};
  q.$\phi$ := true;
  forall $(q', \rightarrow, q) \in T$ do
    if $q'.\text{seenbefore}$ = false then
      q'.seenbefore := true;
      if $q'.\psi_1$ = true then
        L := L \cup {q'}

\[
\begin{array}{|c|c|}
\hline
\text{seenbefore} & L \\
\hline
\emptyset & \{6\} \\
\hline
\end{array}
\]
Case 5: $\phi = E\psi_1 U\psi_2$

marking($\psi_1$);
marking($\psi_2$);
forall $q \in Q$ do
  q.$\phi$:=false;
  q.seenbefore:=false
L:=\emptyset;
forall $q \in Q$ do
  if q.$\psi_2$=true then
    L:=L∪{q}
while $L \neq \emptyset$ do
  pick q from L; L:=L\{q};
  q.$\phi$:=true;
forall (q', _, q) \in T do
  if q'.seenbefore=false then
    q'.seenbefore:=true;
    if q'.$\psi_1$=true then
      L:=L∪{q'}

<table>
<thead>
<tr>
<th>seenbefore</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1}</td>
<td>{1,6}</td>
</tr>
</tbody>
</table>
Case 5: $\phi = E\psi_1 U \psi_2$

marking($\psi_1$);
marking($\psi_2$);
forall $q \in Q$ do
  $q.\phi := false$;
  $q.\text{seenbefore} := false$
$L := \emptyset$;
forall $q \in Q$ do
  if $q.\psi_2 = true$ then
    $L := L \cup \{q\}$
while $L \neq \emptyset$ do
  pick $q$ from $L$; $L := L \backslash \{q\}$;
  $q.\phi := true$;
  forall $(q', \_, q) \in T$ do
    if $q'.\text{seenbefore} = false$ then
      $q'.\text{seenbefore} := true$;
      if $q'.\psi_1 = true$ then
        $L := L \cup \{q'\}$

$\phi = E\text{req}_1 U \text{cs}_1$

<table>
<thead>
<tr>
<th>seenbefore</th>
<th>$L$</th>
</tr>
</thead>
<tbody>
<tr>
<td>${1}$</td>
<td>${6}$</td>
</tr>
</tbody>
</table>
Case 5: $\phi = \mathsf{E}\psi_1 \mathsf{U}\psi_2$

marking($\psi_1$);
marking($\psi_2$);

forall $q \in Q$ do
  q.$\phi$:=$\text{false}$;
  q.seenbefore:=$\text{false}$

L:=$\emptyset$;

forall $q \in Q$ do
  if q.$\psi_2$=$\text{true}$ then
    L:=L$\cup\{q\}$

while L$\neq \emptyset$ do
  pick q from L; L:=L\{q\};
  q.$\phi$:=$\text{true}$;
  forall (q', -, q) $\in T$ do
    if q'.seenbefore=$\text{false}$ then
      q'.seenbefore:=$\text{true}$;
      if q'.$\psi_1$=$\text{true}$ then
        L:=L$\cup\{q'\}$

\[\phi = \mathsf{E}\text{req}_1 \mathsf{U}\text{cs}_1\]
Case 5: $\phi = E\psi_1 U \psi_2$

marking($\psi_1$);
marking($\psi_2$);
forall $q \in Q$ do
  q.\(\phi\) := false;
  q.seenbefore := false
L := \emptyset;
forall $q \in Q$ do
  if $q.\psi_2$ = true then
    L := L \cup \{q\}
while $L \neq \emptyset$ do
  pick q from L; L := L \setminus \{q\};
  q.\(\phi\) := true;
  forall ($q'$, $\_$, $q$) $\in T$ do
    if $q'.\text{seenbefore}$ = false then
      $q'.\text{seenbefore}$ := true;
      if $q'.\psi_1$ = true then
        L := L \cup \{q'\}

<table>
<thead>
<tr>
<th>seenbefore</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>{0, 1, 7}</td>
<td>{6, 7}</td>
</tr>
</tbody>
</table>
Case 5: $\phi = E\psi_1 U \psi_2$

marking($\psi_1$);
marking($\psi_2$);
for all $q \in Q$ do
  $q.\phi := \text{false}$;
  $q.\text{seenbefore} := \text{false}$
$L := \emptyset$;
for all $q \in Q$ do
  if $q.\psi_2 = \text{true}$ then
    $L := L \cup \{q\}$
while $L \neq \emptyset$ do
  pick $q$ from $L$; $L := L \setminus \{q\}$;
  $q.\phi := \text{true}$;
  for all $(q', \_, q) \in T$ do
    if $q'.\text{seenbefore} = \text{false}$ then
      $q'.\text{seenbefore} := \text{true}$;
      if $q'.\psi_1 = \text{true}$ then
        $L := L \cup \{q'\}$

\[ \phi = E\text{req}_1 U \text{cs}_1 \]
Case 5: $\phi = E\psi_1 U \psi_2$

marking($\psi_1$);  
marking($\psi_2$);  
forall $q \in Q$ do  
    $q.\phi := \text{false}$;  
    $q.\text{seenbefore} := \text{false}$  
L := \emptyset;  
forall $q \in Q$ do  
    if $q.\psi_2 = \text{true}$ then  
        L := L \cup \{q\}$  
while $L \neq \emptyset$ do  
    pick $q$ from L; L := L \setminus \{q\};  
    $q.\phi := \text{true}$;  
    forall ($q', \rightarrow, q$) $\in T$ do  
        if $q'.\text{seenbefore} = \text{false}$ then  
            $q'.\text{seenbefore} := \text{true}$;  
            if $q'.\psi_1 = \text{true}$ then  
                L := L \cup \{q'\}  

\[
\phi = E\text{req}_1 U \text{cs}_1
\]

\[
\begin{array}{|c|c|}
\hline
\text{seenbefore} & L \\
\hline
\{0,1,3,7\} & \{3,7\} \\
\hline
\end{array}
\]
Case 5: $\phi = E\psi_1 U \psi_2$

marking($\psi_1$);
marking($\psi_2$);

forall $q \in Q$ do
  $q.\phi := \text{false}$;
  $q.\text{seenbefore} := \text{false}$

$L := \emptyset$;

forall $q \in Q$ do
  if $q.\psi_2 = \text{true}$ then
    $L := L \cup \{q\}$

while $L \neq \emptyset$ do
  pick $q$ from $L$; $L := L \setminus \{q\}$;
  $q.\phi := \text{true}$;
  forall $(q', \ldots, q) \in T$ do
    if $q'.\text{seenbefore} = \text{false}$ then
      $q'.\text{seenbefore} := \text{true}$; $q'.\psi_1 := \text{true}$ then
      $L := L \cup \{q'\}$

<table>
<thead>
<tr>
<th>seenbefore</th>
<th>$L$</th>
</tr>
</thead>
<tbody>
<tr>
<td>${0,1,3,4,7}$</td>
<td>${3,7}$</td>
</tr>
</tbody>
</table>
Case 5: $\phi = E\psi_1 U \psi_2$

marking($\psi_1$);
marking($\psi_2$);

forall $q \in Q$ do
  q.\phi:=false;
  q.seenbefore:=false
L:=\emptyset;

forall $q \in Q$ do
  if $q.\psi_2=\text{true}$ then
    L:=L\cup\{q\}

while $L \neq \emptyset$ do
  pick q from L; L:=L\\{q\};
  q.\phi:=true;

forall $(q', , q) \in T$ do
  if q'.seenbefore=\text{false} then
    q'.seenbefore:=\text{true};
    if q'.\psi_1=\text{true} then
      L:=L\cup\{q'\}

\begin{tabular}{|c|c|}
  \hline
  \text{seenbefore} & L \\
  \hline
  \{0,1,3,4,7\} & \{7\} \\
  \hline
\end{tabular}
Case 5: $\phi = E\psi_1 U\psi_2$

```
marking($\psi_1$);
marking($\psi_2$);
for all $q \in Q$ do
    q.$\phi$ := false;
    q.seenbefore := false
L := {};
for all $q \in Q$ do
    if q.$\psi_2$ = true then
        L := L $\cup$ {q}
while L $\neq$ {} do
    pick q from L; L := L \{q};
    q.$\phi$ := true;
for all $(q', \rightarrow, q) \in T$ do
    if q'.seenbefore = false then
        q'.seenbefore := true;
        if q'.$\psi_1$ = true then
            L := L $\cup$ {q'}
```

<table>
<thead>
<tr>
<th>seenbefore</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>{0,1,2,3,4,7}</td>
<td>{7}</td>
</tr>
</tbody>
</table>
Case 5: $\phi = E\psi_1 U \psi_2$

marking($\psi_1$);
marking($\psi_2$);
forall $q \in Q$ do
  $q.\phi := false$;
  $q.seenbefore := false$
L := $\emptyset$;
forall $q \in Q$ do
  if $q.\psi_2 = true$ then
    $L := L \cup \{q\}$
while $L \neq \emptyset$ do
  pick $q$ from $L$; $L := L \\setminus \{q\}$;
  $q.\phi := true$;
forall $(q',\_, q) \in T$ do
  if $q'.seenbefore = false$ then
    $q'.seenbefore := true$;
  if $q'.\psi_1 = true$ then
    $L := L \cup \{q'\}$

\[ \phi = Ereq_1 U cs_1 \]
Case 5: $\phi = E\psi_1 U \psi_2$

marking($\psi_1$);
marking($\psi_2$);

forall $q \in Q$ do

  $q.\phi := false$;
  $q.seenbefore := false$

$L := \emptyset$;

forall $q \in Q$ do

  if $q.\psi_2 = true$ then
    $L := L \cup \{q\}$

while $L \neq \emptyset$ do

  pick $q$ from $L$; $L := L \setminus \{q\}$;
  $q.\phi := true$;

forall $(q', \_ , q) \in T$ do

  if $q'.seenbefore = false$ then
    $q'.seenbefore := true$;
    if $q'.\psi_1 = true$ then
      $L := L \cup \{q'\}$


<table>
<thead>
<tr>
<th>seenbefore</th>
<th>$L$</th>
</tr>
</thead>
<tbody>
<tr>
<td>${0,1,2,3,4,5,7}$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>
Case 6: \( \phi = A\psi_1 U \psi_2 \)

marking(\(\psi_1\));
marking(\(\psi_2\));
L:=\(\emptyset\);
for all \(q \in Q\) do
  q.nb:=degree(q);
  q.\(\phi\):=false
for all \(q \in Q\) do
  if q.\(\psi_2\)=true then
    L:=L\(\cup\){q}
while L\(\neq\emptyset\) do
  pick q from L; L:=L\(\setminus\){q};
  q.\(\phi\):=true;
for all (\(q',\_\,\_\,\_\), q) \(\in T\) do
  q'.nb:=q'.nb - 1;
  if q'.nb=0 and q'.\(\psi_1\)=true and q'.\(\phi\)=false then
    L:=L\(\cup\){q'}

\(\phi = Areq_1 U cs_1\)

- **nb**
  - 0 1 2 3 4 5 6 7
  - **L**
Case 6: $\phi = A\psi_1 U \psi_2$

marking($\psi_1$);
marking($\psi_2$);
L:=\emptyset;

forall q ∈ Q do
    q.nb:=degree(q);
    q.$\phi$:=false

forall q ∈ Q do
    if q.$\psi_2$=true then
        L:=L∪{q}

while L\neq \emptyset do
    pick q from L; L:=L\{q};
    q.$\phi$:=true;
    forall (q', _, q) ∈ T do
        q'.nb:=q'.nb - 1;
        if q'.nb=0 and q'.$\psi_1$=true and q'.$\phi$=false then
            L:=L∪{q'}

$\phi = Areq_1 U cs_1$
Case 6: $\phi = A \psi_1 U \psi_2$

marking($\psi_1$);
marking($\psi_2$);
L:=∅;
for all $q \in Q$ do
    q.nb:=degree(q);
    q.φ:=false
for all $q \in Q$ do
    if q.ψ_2=true then
        L:=L∪{q}
while L ≠ ∅ do
    pick q from L; L:=L\{q};
    q.φ:=true;
for all $(q',_,q) \in T$ do
    q'.nb:=q'.nb - 1;
    if q'.nb=0 and q'.ψ_1=true and q'.φ=false then
        L:=L∪{q'}
Case 6: $\phi = A\psi_1 U \psi_2$

marking($\psi_1$);
marking($\psi_2$);
$L:=\emptyset$;

forall $q \in Q$ do
  q.nb:=degree(q);
  q.$\phi$:=false

forall $q \in Q$ do
  if q.$\psi_2$=true then
    L:=L∪{q}

while $L \neq \emptyset$ do
  pick q from L; L:=L\{q};
  q.$\phi$:=true;
  forall $(q',\_\_\_\_, q) \in T$ do
    q'.nb:=q'.nb - 1;
    if q'.nb=0 and q'.$\psi_1$=true and q'.$\phi$=false then
      L:=L∪{q'}

\[
\phi = Areq_1 U cs_1
\]

<table>
<thead>
<tr>
<th>nb</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7</td>
<td>2 2 2 2 2 2 1 1</td>
</tr>
</tbody>
</table>
Case 6: $\phi = A\psi_1 U \psi_2$

marking($\psi_1$);
marking($\psi_2$);
$L := \emptyset$;

forall $q \in Q$ do
  q.nb := degree(q);
  q.\phi := false

forall $q \in Q$ do
  if $q.\psi_2 = \text{true}$ then
    $L := L \cup \{q\}$

while $L \neq \emptyset$ do
  pick q from L; $L := L \setminus \{q\}$;
  q.\phi := true;
  forall $(q', -, q) \in T$ do
    q'.nb := q'.nb - 1;
    if q'.nb = 0 and q'.\psi_1 = \text{true}
    and q'.\phi = \text{false} then
      $L := L \cup \{q'\}$

$L := \{4, 6\}$
Case 6: $\phi = A\psi_1 U \psi_2$

marking($\psi_1$);
marking($\psi_2$);
$L := \emptyset$;

forall $q \in Q$ do
  q.nb := degree(q);
  q.$\phi$ := false

forall $q \in Q$ do
  if q.$\psi_2$ = true then
    L := L \cup \{q\}

while $L \neq \emptyset$ do
  pick q from L; L := L \{q\};
  q.$\phi$ := true;
  forall $(q', \_ , q) \in T$ do
    q'.nb := q'.nb - 1;
    if q'.nb = 0 and q'.$\psi_1$ = true and q'.$\phi$ = false then
      L := L \cup \{q'\}

\[ \phi = Areq_1 U cs_1 \]
Case 6: $\phi = \text{A} \psi_1 \text{U} \psi_2$

marking($\psi_1$);
marking($\psi_2$);
L:=∅;
forall $q \in Q$ do
  q.nb:=degree(q);
  q.\phi:=false
forall $q \in Q$ do
  if q.\psi_2=true then
    L:=L∪{q}
while L≠∅ do
  pick q from L; L:=L\{q};
  q.\phi:=true;
forall ($q'$, _, q) ∈ T do
  q'.nb:=q'.nb - 1;
  if q'.nb=0 and q'.\psi_1=true and q'.\phi=false then
    L:=L∪ {q'}
Case 6: $\phi = A\psi_1 U \psi_2$

marking($\psi_1$);
marking($\psi_2$);
L:=\emptyset;
forall q \in Q do
  q.nb:=degree(q);
  q.\phi:=false
forall q \in Q do
  if q.\psi_2=true then
    L:=L\cup\{q\}
while L\neq \emptyset do
  pick q from L; L:=L\{q\};
  q.\phi:=true;
forall (q',_,q) \in T do
  q'.nb:=q'.nb - 1;
  if q'.nb=0 and q'.\psi_1=true and q'.\phi=false then
    L:=L\cup \{q'\}
Case 6: $\phi = A\psi_1 U \psi_2$

<table>
<thead>
<tr>
<th>$\phi = Areq_1 U cs_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Diagram" /></td>
</tr>
</tbody>
</table>

marking($\psi_1$);
marking($\psi_2$);
L:=\emptyset;
forall $q \in Q$ do
    q.nb:=degree(q);
    q.$\phi$:=false
forall $q \in Q$ do
    if q.$\psi_2$=true then
        L:=L∪\{q\}
while L\neq \emptyset do
    pick q from L; L:=L\{q\};
    q.$\phi$:=true;
forall ($q', \_, q$) $\in T$ do
    q'.$nb$:=q'.$nb$ - 1;
    if q'.$nb$=0 and q'.$\psi_1$=true and q'.$\phi$=false then
        L:=L∪\{q’\}
Case 6: $\phi = A\psi_1 U \psi_2$

marking($\psi_1$);
marking($\psi_2$);
L:=∅;
forall q ∈ Q do
    q.nb:=degree(q);
    q.$\phi$:=false
forall q ∈ Q do
    if q.$\psi_2$=true then
        L:=L∪{q}
while L≠∅ do
    pick q from L; L:=L\{q};
    q.$\phi$:=true;
forall (q′, _, q) ∈ T do
    q′.nb:=q′.nb - 1;
    if q′.nb=0 and q′.$\psi_1$=true and q′.$\phi$=false then
        L:=L∪{q′}

$\phi = Areq_1 U cs_1$

<table>
<thead>
<tr>
<th>nb</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7</td>
<td>∅</td>
</tr>
<tr>
<td>2 1 2 1 1 2 1 1</td>
<td></td>
</tr>
</tbody>
</table>
Exercises

Check $AG(EF(idle_1 \land idle_2))$
Exercises

Check $\text{AG}(\text{EF}(<idle_1 \land idle_2>))$

$$\text{AG}(\text{EF}(<idle_1 \land idle_2>))$$

$$\equiv \neg (E\text{trueU}\neg(\text{EF}(<idle_1 \land idle_2>)))$$

$$\equiv \neg (E\text{trueU}(\text{E}(\text{trueU}(<idle_1 \land idle_2>))))$$
Check $\text{AG}(\text{EF}(\text{idle}_1 \land \text{idle}_2))$

- $\text{AG}(\text{EF}(\text{idle}_1 \land \text{idle}_2))$
  - $\equiv \neg (E \text{true} U \neg (\text{EF}(\text{idle}_1 \land \text{idle}_2)))$
  - $\equiv \neg (E \text{true} U \neg (E(\text{true} U (\text{idle}_1 \land \text{idle}_2))))$

- mark $\text{idle}_1$: states 0, 2, 5
Exercises

Check $A\Gamma(E\forall(idle_1 \land idle_2))$

$A\Gamma(E\forall(idle_1 \land idle_2))$
- $\equiv \neg(Etrue \cup \neg(E\forall(idle_1 \land idle_2)))$
- $\equiv \neg(Etrue \cup (E(true \cup (idle_1 \land idle_2))))$

- mark $idle_1$: states 0, 2, 5
- mark $idle_2$: states 0, 1, 4
- case 3, mark $idle_1 \land idle_2$: state 0
Check $AG(EF(idle_1 \land idle_2))$

$AG(EF(idle_1 \land idle_2))$

$\equiv \neg (E(trueU\neg(EF(idle_1 \land idle_2))))$

$\equiv \neg (E(trueU(E(falseU(idle_1 \land idle_2))))))$

- mark $idle_1$: states 0, 2, 5
- mark $idle_2$: states 0, 1, 4
- case 3, mark $idle_1 \land idle_2$: state 0
- case 5, mark $\phi_1 = E(trueU(idle_1 \land idle_2))$
Exercises

Check \( \text{AG}(\text{EF}(\text{idle}_1 \land \text{idle}_2)) \)

\[
\text{AG}(\text{EF}(\text{idle}_1 \land \text{idle}_2)) \\
\equiv \neg (E(\text{true} U \neg (\text{EF}(\text{idle}_1 \land \text{idle}_2)))) \\
\equiv \neg (E(\text{true} U \neg (E(\text{true} U (\text{idle}_1 \land \text{idle}_2)))))
\]

- mark \( \text{idle}_1 \): states 0, 2, 5
- mark \( \text{idle}_2 \): states 0, 1, 4
- case 3, mark \( \text{idle}_1 \land \text{idle}_2 \): state 0
- case 5, mark \( \phi_1 = E(\text{true} U (\text{idle}_1 \land \text{idle}_2)) \)
Check $\text{AG(} \text{EF}(idle_1 \land idle_2))$
Exercises

Check $\text{AG}(\text{EF}(\text{idle}_1 \land \text{idle}_2))$

$\text{AG}(\text{EF}(\text{idle}_1 \land \text{idle}_2))$

- $\equiv \neg (\text{E} \text{true} \text{U} \neg (\text{EF}(\text{idle}_1 \land \text{idle}_2)))$
- $\equiv \neg (\text{E} \text{true} \text{U} (\text{E}(\text{true} \text{U}(\text{idle}_1 \land \text{idle}_2))))$

- mark $\text{idle}_1$: states 0, 2, 5
- mark $\text{idle}_2$: states 0, 1, 4
- case 3, mark $\text{idle}_1 \land \text{idle}_2$: state 0
- case 5, mark $\phi_1 = \text{E}(\text{true} \text{U}(\text{idle}_1 \land \text{idle}_2))$
Exercises

Check $\text{AG}(\text{EF}(idle_1 \land idle_2))$

$\text{AG}(\text{EF}(idle_1 \land idle_2))$

- $\equiv \neg (\text{E}(true \text{U} \neg (\text{EF}(idle_1 \land idle_2))))$
- $\equiv \neg (\text{E}(true \text{U} (\text{E}(true \text{U} (idle_1 \land idle_2)))))$

- mark $idle_1$: states 0, 2, 5
- mark $idle_2$: states 0, 1, 4
- case 3, mark $idle_1 \land idle_2$: state 0
- case 5, mark $\phi_1 = \text{E}(true \text{U} (idle_1 \land idle_2))$
Exercises

Check AG(EF(idle₁ ∧ idle₂))

AG(EF(idle₁ ∧ idle₂))

≡ ¬(EtrueU¬(EF(idle₁ ∧ idle₂)))
≡ ¬(EtrueU¬(E(trueU(idle₁ ∧ idle₂))))

- mark idle₁: states 0, 2, 5
- mark idle₂: states 0, 1, 4
- case 3, mark idle₁ ∧ idle₂: state 0
- case 5, mark φ₁ = E(trueU(idle₁ ∧ idle₂))
Exercises

Check $\text{AG(\text{EF}(idle_1 \land idle_2))}$

$\text{AG(\text{EF}(idle_1 \land idle_2))}$

$\equiv \neg(\text{E(trueU}\neg(\text{EF}(idle_1 \land idle_2))))$

$\equiv \neg(\text{E(trueU}(\text{E(trueU}(idle_1 \land idle_2))))))$

- mark $idle_1$: states 0, 2, 5
- mark $idle_2$: states 0, 1, 4
- case 3, mark $idle_1 \land idle_2$: state 0
- case 5, mark $\phi_1 = E(\text{trueU}(idle_1 \land idle_2))$
Exercises

Check \(AG(EF(idle_1 \land idle_2))\)

\[
AG(EF(idle_1 \land idle_2)) = \neg (E\text{true}U\neg(EF(idle_1 \land idle_2)))
\]

- mark \(idle_1\): states 0, 2, 5
- mark \(idle_2\): states 0, 1, 4
- case 3, mark \(idle_1 \land idle_2\): state 0
- case 5, mark \(\phi_1 = E(\text{true}U(idle_1 \land idle_2))\)
Exercises

Check AG(\(EF(idle_1 \land idle_2)\))

\[
AG(\(EF(idle_1 \land idle_2)\)) \\
\equiv \neg (E(true U \neg (EF(idle_1 \land idle_2)))) \\
\equiv \neg (E(true U (E(true U (idle_1 \land idle_2)))))
\]

- mark \(idle_1\): states 0, 2, 5
- mark \(idle_2\): states 0, 1, 4
- case 3, mark \(idle_1 \land idle_2\): state 0
- case 5, mark \(\phi_1 = E(true U (idle_1 \land idle_2))\)
Check $\text{AG}(\text{EF}(idle_1 \land idle_2))$

$\text{AG}(\text{EF}(idle_1 \land idle_2))$

\[\equiv \neg(\text{E}\text{true} \text{U} \neg(\text{EF}(idle_1 \land idle_2)))\]

\[\equiv \neg(\text{E}\text{true} \text{U} \neg(\text{E}(\text{true} \text{U}(idle_1 \land idle_2))))\]

- mark $idle_1$: states 0, 2, 5
- mark $idle_2$: states 0, 1, 4
- case 3, mark $idle_1 \land idle_2$: state 0
- case 5, mark $\phi_1 = \text{E}(\text{true} \text{U}(idle_1 \land idle_2))$
Exercises

Check $\text{AG}(\text{EF}(idle_1 \land idle_2))$

AG($\text{EF}(idle_1 \land idle_2)$)

$\equiv \lnot (E(\text{true} U \lnot (\text{EF}(idle_1 \land idle_2))))$

$\equiv \lnot (E(\text{true} U (E(\text{true} U (idle_1 \land idle_2))))))$

- mark $idle_1$: states 0, 2, 5
- mark $idle_2$: states 0, 1, 4
- case 3, mark $idle_1 \land idle_2$: state 0
- case 5, mark $\phi_1 = E(\text{true} U (idle_1 \land idle_2))$
Check $\text{AG}(\text{EF}(idle_1 \land idle_2))$

$\text{AG}(\text{EF}(idle_1 \land idle_2))$

$\equiv \neg (E(\text{true} U \neg (\text{EF}(idle_1 \land idle_2))))$

$\equiv \neg (E(\text{true} U (\text{E}(\text{true} U (idle_1 \land idle_2))))))$

- mark $idle_1$: states 0, 2, 5
- mark $idle_2$: states 0, 1, 4
- case 3, mark $idle_1 \land idle_2$: state 0
- case 5, mark $\phi_1 = E(\text{true} U (idle_1 \land idle_2))$
- case 2, mark $\phi_2 = \neg \phi_1$
Exercises

Check $AG(EF(idle_1 \land idle_2))$

$AG(EF(idle_1 \land idle_2))$

- $\equiv \neg(E(\text{true} U \neg(EF(idle_1 \land idle_2))))$
- $\equiv \neg(E(\text{true} U (E(\text{true} U (idle_1 \land idle_2))))))$

- mark $idle_1$: states 0, 2, 5
- mark $idle_2$: states 0, 1, 4
- case 3, mark $idle_1 \land idle_2$: state 0
- case 5, mark $\phi_1 = E(\text{true} U (idle_1 \land idle_2))$
- case 2, mark $\phi_2 = \neg \phi_1$
Check $\text{AG}(\text{EF}(idle_1 \land idle_2))$

$\text{AG}(\text{EF}(idle_1 \land idle_2))$

- $\equiv \neg(\text{E}true \lor \neg(\text{EF}(idle_1 \land idle_2)))$
- $\equiv \neg(\text{E}true \lor \neg(\text{E}(true \lor \neg(\text{EF}(idle_1 \land idle_2)))))$

- mark $idle_1$: states 0, 2, 5
- mark $idle_2$: states 0, 1, 4
- case 3, mark $idle_1 \land idle_2$: state 0
- case 5, mark $\phi_1 = \text{E}(true \lor \neg(\text{EF}(idle_1 \land idle_2)))$
- case 2, mark $\phi_2 = \neg\phi_1$
- case 5, mark $\phi_3 = \text{E}(true \lor \phi_2)$
Check $\text{AG}(\text{EF}(idle_1 \land idle_2))$

$\text{AG}(\text{EF}(idle_1 \land idle_2))$

$\equiv \neg(\text{E}\text{trueU}\neg(\text{EF}(idle_1 \land idle_2)))$

$\equiv \neg(\text{E}\text{trueU}(\text{E}(\text{trueU}(idle_1 \land idle_2))))$

- mark $idle_1$: states 0, 2, 5
- mark $idle_2$: states 0, 1, 4
- case 3, mark $idle_1 \land idle_2$: state 0
- case 5, mark $\phi_1 = \text{E}(\text{trueU}(idle_1 \land idle_2))$
- case 2, mark $\phi_2 = \neg \phi_1$
- case 5, mark $\phi_3 = \text{E}(\text{trueU}\phi_2)$
Check $AG(\text{EF}(idle_1 \land idle_2))$

$AG(\text{EF}(idle_1 \land idle_2))$

- $\equiv \neg (E\text{true}U\neg(\text{EF}(idle_1 \land idle_2)))$
- $\equiv \neg (E\text{true}U(\neg(\text{true}U(idle_1 \land idle_2))))$

- mark $idle_1$: states 0, 2, 5
- mark $idle_2$: states 0, 1, 4
- case 3, mark $idle_1 \land idle_2$: state 0
- case 5, mark $\phi_1 = E(\text{true}U(idle_1 \land idle_2))$
- case 2, mark $\phi_2 = \neg \phi_1$
- case 5, mark $\phi_3 = E(\text{true}U\phi_2)$
- case 2, mark $\phi_4 = \neg \phi_3$
Check \( \text{AG}(\text{EF}(idle_1 \land idle_2)) \)

\[
\text{AG}(\text{EF}(idle_1 \land idle_2)) \equiv \neg (E(\text{true} \text{U} \neg (\text{EF}(idle_1 \land idle_2)))) \\
\equiv \neg (E(\text{true} \text{U} (E(\text{true} \text{U} (idle_1 \land idle_2)))))
\]

- mark \( idle_1 \): states 0, 2, 5
- mark \( idle_2 \): states 0, 1, 4
- case 3, mark \( idle_1 \land idle_2 \): state 0
- case 5, mark \( \phi_1 = E(\text{true} \text{U} (idle_1 \land idle_2)) \)
- case 2, mark \( \phi_2 = \neg \phi_1 \)
- case 5, mark \( \phi_3 = E(\text{true} \text{U} \phi_2) \)
- case 2, mark \( \phi_4 = \neg \phi_3 \)
Exercise

marking($\psi_1$);
marking($\psi_2$);
L:=\emptyset;

forall $q \in Q$ do
  q.nb:=degree(q);
  q.$\phi$:=false

forall $q \in Q$ do
  if $q.\psi_2$=true then
    L:=L∪{q}

while $L \neq \emptyset$ do
  pick q from L; L:=L\{q};
  q.$\phi$:=true;
  forall ($q'$, _, q) ∈ $T$ do
    q'.nb:=q'.nb - 1;
    if q'.nb=0 and q'.$\psi_1$=true and q'.$\phi$=false then
      L:=L∪{q'}

$\phi = Areq_1 U cs_2$
Exercise

marking($\psi_1$);
marking($\psi_2$);
L:=\emptyset;
forall q \in Q do
    q.nb:=degree(q);
    q.\phi:=false
forall q \in Q do
    if q.\psi_2=true then
        L:=L\cup\{q\}
while L\neq \emptyset do
    pick q from L; L:=L\\{q\};
    q.\phi:=true;
forall (q',_,q) \in T do
    q'.nb:=q'.nb - 1;
    if q'.nb=0 and q'.\psi_1=true and q'.\phi=false then
        L:=L\cup\{q'\}

$\phi = Areq_1 Ucs_2$
Exercise

marking(ψ₁);
marking(ψ₂);
L:=∅;
forall q ∈ Q do
  q.nb:=degree(q);
  q.φ:=false
forall q ∈ Q do
  if q.ψ₂=true then
    L:=L∪{q}
while L≠∅ do
  pick q from L; L:=L\{q};
  q.φ:=true;
forall (q',_,q) ∈ T do
  q'.nb:=q'.nb - 1;
  if q'.nb=0 and q'.ψ₁=true and q'.φ=false then
    L:=L∪ {q'}
Exercise

marking($\psi_1$);
marking($\psi_2$);
L:=∅;
forall q ∈ Q do
    q.nb:=degree(q);
    q.φ:=false
forall q ∈ Q do
    if q.ψ_2=true then
        L:=L∪{q}
while L≠∅ do
    pick q from L; L:=L\{q};
    q.φ:=true;
forall (q′, , q) ∈ T do
    q′.nb:=q′.nb - 1;
    if q′.nb=0 and q′.ψ_1=true and q′.φ=false then
        L:=L∪{q′}

ϕ = Areq_1 Ucs_2

<table>
<thead>
<tr>
<th>nb</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2 2</td>
</tr>
<tr>
<td>1</td>
<td>2 2</td>
</tr>
<tr>
<td>2</td>
<td>2 2</td>
</tr>
<tr>
<td>3</td>
<td>2 2</td>
</tr>
<tr>
<td>4</td>
<td>2 2</td>
</tr>
<tr>
<td>5</td>
<td>2 2</td>
</tr>
<tr>
<td>6</td>
<td>2 2</td>
</tr>
<tr>
<td>7</td>
<td>2 2</td>
</tr>
</tbody>
</table>

$\emptyset$
Exercise

marking(ψ₁);
marking(ψ₂);
L:=∅;
forall q ∈ Q do
  q.nb:=degree(q);
  q.φ:=false
forall q ∈ Q do
  if q.ψ₂=true then
    L:=L∪{q}
while L ≠ ∅ do
  pick q from L; L:=L\{q};
  q.φ:=true;
forall (q’, _, q) ∈ T do
  q’.nb:=q’.nb - 1;
  if q’.nb=0 and q’.ψ₁=true and q’.φ=false then
    L:=L∪{q’}

φ = Areq₁ U cs₂

<table>
<thead>
<tr>
<th>nb</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1 2 3 4 5 6 7</td>
</tr>
<tr>
<td>2</td>
<td>2 2 2 2 2 2 1 1</td>
</tr>
<tr>
<td>7</td>
<td>{5,7}</td>
</tr>
</tbody>
</table>
Exercise

marking($\psi_1$);
marking($\psi_2$);
L:=∅;
forall q ∈ Q do
    q.nb:=degree(q);
    q.φ:=false
forall q ∈ Q do
    if q.ψ_2=true then
        L:=L∪{q}
while L≠∅ do
    pick q from L; L:=L\{q};
    q.φ:=true;
forall (q′, _, q) ∈ T do
    q′.nb:=q′.nb - 1;
    if q′.nb=0 and q′.ψ_1=true and q′.φ=false then
        L:=L∪{q′}

$\phi = \text{A} req_1 \text{U} cs_2$
Exercise

marking($\psi_1$);
marking($\psi_2$);
L:=\emptyset;
forall q \in Q do
  q.nb:=\text{degree}(q);
  q.\phi:=\text{false}
forall q \in Q do
  if q.\psi_2=\text{true} then
    L:=L\cup\{q\}
while L\neq \emptyset do
  pick q from L; L:=L\setminus\{q\};
  q.\phi:=\text{true};
forall (q',\_, q) \in T do
  q'.nb:=q'.nb - 1;
  if q'.nb=0 and q'.\psi_1=\text{true} and q'.\phi=\text{false} then
    L:=L \cup \{q'\}

$\phi = Areq_1 U cs_2$

<table>
<thead>
<tr>
<th>nb</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7</td>
<td>2 2 1 2 2 2 1 1</td>
</tr>
</tbody>
</table>
Exercise

marking($\psi_1$);
marking($\psi_2$);
L:=\emptyset;

forall q \in Q do
|  q.nb:=\deg(q);
|  q.\phi:=false

forall q \in Q do
|  if q.\psi_2=true then
|  |  L:=L\cup\{q\}

while L\neq \emptyset do
|  pick q from L; L:=L\setminus\{q\};
|  q.\phi:=true;
|  forall (q',\_, q) \in T do
|  |  q'.nb:=q'.nb - 1;
|  |  if q'.nb=0 and q'.\psi_1=true and q'.\phi=false then
|  |  |  L:=L\cup\{q'\}

$\phi = Areq_1 U cs_2$

\[
\begin{array}{c|c}
\text{nb} & L \\
0 & 1 2 3 4 5 6 7 \\
2 & 2 1 2 2 2 1 1 \\
\end{array}
\]
Exercise

marking($\psi_1$);
marking($\psi_2$);
L:=\emptyset;
forall q \in Q do
  q.nb:=degree(q);
  q.\phi:=false
forall q \in Q do
  if q.\psi_2=true then
    L:=L\cup\{q\}
while L\neq \emptyset do
  pick q from L; L:=L\{q\};
  q.\phi:=true;
forall (q', \_, q) \in T do
  q'.nb:=q'.nb - 1;
  if q'.nb=0 and q'.\psi_1=true and q'.\phi=false then
    L:=L\cup \{q'\}

$\phi = A\text{req}_1 U c\text{cs}_2$

<table>
<thead>
<tr>
<th>nb</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7</td>
<td>2 2 1 2 2 1 1 1</td>
</tr>
</tbody>
</table>
Exercise

marking($\psi_1$);
marking($\psi_2$);
$L:=\emptyset$;

forall $q \in Q$ do

  q.nb:=degree($q$);
  q.$\phi$:=false

forall $q \in Q$ do

  if q.$\psi_2$=true then
    L:=L∪{q}

while $L \neq \emptyset$ do

  pick q from L; L:=L\{q};
  q.$\phi$:=true;

forall $(q',\_, q) \in T$ do

  q'.nb:=q'.nb - 1;
  if q'.nb=0 and q'.$\psi_1$=true and q'.$\phi$=false then
    L:=L∪{q'}

---

$\phi = Areq_1 Ucs_2$

---

\[
\begin{array}{c|cccccc}
   & nb & L \\
\hline
0 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
2 & 2 & 1 & 1 & 2 & 1 & 1 & 1 & \emptyset
\end{array}
\]
LTL model-checking

Algorithm working on path formulae

Principle for checking if $\mathcal{A} \models \phi$

1. construct automaton $\mathcal{B}_{\neg \phi}$ recognising all executions not satisfying $\phi$
2. construct the synchronised product $\mathcal{A} \otimes \mathcal{B}_{\neg \phi}$
3. if its recognised language is empty, then $\mathcal{A} \models \phi$
Example

\begin{align*}
\mathcal{A} & \\
\mathcal{B}_{\neg\phi} \text{ for } \phi = G(P \implies XFQ)
\end{align*}
Example

$A$

$B_{\neg \phi}$ for $\phi = G(P \implies XFQ)$

$A \otimes B_{\neg \phi}$
Example

\( A \)

\[\begin{array}{cccccc}
\neg P & \neg Q & \neg P & \rightarrow & \neg Q & t_1 \\
\neg P & \neg Q & & & & t_4 \\
\neg P & \neg Q & & & & t_5 \\
\end{array}\]

\[\begin{array}{cc}
P & Q \\
\neg P & \neg Q \\
\neg P & \neg Q \\
\end{array}\]

\( B_{\neg \phi} \) for \( \phi = G(P \implies XFQ) \)

\[\begin{array}{cccccc}
q_0 & & & & & q_1 \\
& \downarrow & & & & \downarrow u_1: P, Q \\
& P, Q & & & & \neg P, \neg Q \\
& P, \neg Q & & & & \neg P, Q \\
& \neg P, Q & & & & \neg P, \neg Q \\
& \neg P, \neg Q & & & & \\
\end{array}\]

\( A \otimes B_{\neg \phi} \)

\[\begin{array}{cccccc}
\neg P & \neg Q & \neg P & \rightarrow & \neg Q & t_1 \otimes u_0 \\
\neg P & \neg Q & & & & t_2 \oplus u_0 \\
\neg P & \neg Q & & & & \\
\end{array}\]
Example

**Example**

$$A$$

$$B_{\neg \phi}$$ for $$\phi = G(P \Rightarrow XFQ)$$

$$A \otimes B_{\neg \phi}$$
Example

\[ A \]

\[ \neg P, \neg Q \]

\[ \neg P, Q \]

\[ \neg P, \neg Q \]

\[ P, Q \]

\[ P \to Q \]

\[ B_{\neg \phi} \text{ for } \phi = G(\neg P \to \neg Q) \]

\[ B_{\neg \phi} \]

\[ q_0 \rightarrow u_1: P, Q \]

\[ u_0: \neg P, \neg Q \]

\[ q_0 \rightarrow q_1 \]

\[ u_2: P, \neg Q \]

\[ u_1: P, Q \]

\[ u_2: P, \neg Q \]

\[ A \otimes B_{\neg \phi} \]

\[ \neg P \]

\[ \neg Q \]

\[ \neg P, \neg Q \]

\[ \neg P, Q \]

\[ P, Q \]

\[ \neg P, \neg Q \]

\[ \neg P \]

\[ \neg Q \]
Example

\[ A \]

\[ B_{\neg \phi} \text{ for } \phi = G(P \implies XFQ) \]

\[ A \otimes B_{\neg \phi} \]
Example

### $A$

- $\neg P$, $\neg Q$
- $P$, $Q$
- $\neg P$, $\neg Q$
- $\neg P$, $\neg Q$
- $\neg P$, $\neg Q$

### $B_{\neg \phi}$ for $\phi = G(P \implies XFQ)$

- $q_0$: $P$, $Q$
- $u_0$: $\neg P$, $Q$
- $u_1$: $P$, $Q$
- $u_2$: $P$, $\neg Q$

### $A \otimes B_{\neg \phi}$

- $\neg P$, $\neg Q$
- $P$, $Q$
- $\neg P$, $\neg Q$
- $\neg P$, $\neg Q$
- $\neg P$, $\neg Q$
Example

$A$

$B_{\neg \phi}$ for $\phi = G(P \implies XFQ)$

$A \otimes B_{\neg \phi}$
Example

\[ \mathcal{A} \]

\[ \mathcal{B}_{\neg \phi} \text{ for } \phi = G(P \implies XFQ) \]

\[ \mathcal{A} \otimes \mathcal{B}_{\neg \phi} \]

\[ \mathcal{A} \not\models \phi \]

Laure Petrucci

Advanced modelling techniques
Exercise

\[ A \]

\[ B_{\neg \phi} \text{ for } \phi = G\neg (cs_1 \land cs_2) \]
Exercise

\[ A \]

For \( \phi = G\neg (cs_1 \land cs_2) \)

\[ \neg \phi \equiv \neg G\neg (cs_1 \land cs_2) \]

\[ \equiv \neg (\neg F\neg (cs_1 \land cs_2)) \]

\[ \equiv F(cs_1 \land cs_2) \]

\[ \equiv \text{trueU}(cs_1 \land cs_2) \]
Exercise

A


Bφ for φ = G¬(cs₁ ∧ cs₂)

¬φ ≡ ¬G¬(cs₁ ∧ cs₂)
≡ ¬(¬F¬(cs₁ ∧ cs₂))
≡ F(cs₁ ∧ cs₂)
≡ trueU(cs₁ ∧ cs₂)
Exercise

\[ A \]

\[ B_{\neg \phi} \text{ for } \phi = G\neg(cs_1 \land cs_2) \]

\[
\neg \phi \equiv \neg G\neg(cs_1 \land cs_2) \\
\equiv \neg (\neg F\neg(cs_1 \land cs_2)) \\
\equiv F(cs_1 \land cs_2) \\
\equiv \text{true}U(cs_1 \land cs_2)
\]

\[ A \otimes B_{\neg \phi} \]
Exercise

\(\mathcal{A} \otimes \mathcal{B}_{\neg \phi}\)

All transitions of \(\mathcal{A}\) synchronise with \(u_0\).
So there is no accepting state and the formula is true.
4 Symbolic model-checking
- Computation of state sets
- Binary Decision Diagrams
- Automata representation
Motivation for symbolic approaches

- state space explosion problem
  - main obstacle with model-checking algorithms
  - because of the necessity to construct the state space
- represent symbolically states and transitions
- it aims at representing concisely large sets of states
Symbolic computation of state sets

Let $\mathcal{A} = \langle Q, E, T, q_0, I, F \rangle$ be an automaton, and $S \subseteq Q$ a set of its states. Let $\phi$ be a CTL formula.

**Notations**

- $\text{Pre}(S) = \{ q \in Q \mid (q, -, q') \in T \land q' \in S \}$ is the set of immediate predecessors of states in $S$
- $\text{Sat}(\phi) = \{ q \in Q \mid q \models \phi \}$ is the set of states of the automaton satisfying formula $\phi$
- $\text{Pre}^*(S)$ is the set of predecessors of states in $S$, whatever the number of steps
Computing $\text{Sat}(\phi)$

\[
\begin{align*}
\text{Sat}(\neg \phi) &= Q \setminus \text{Sat}(\phi) \\
\text{Sat}(\psi_1 \land \psi_2) &= \text{Sat}(\psi_1) \cap \text{Sat}(\psi_2) \\
\text{Sat}(\text{EX}\phi) &= \text{Pre}(\text{Sat}(\phi)) \\
\text{Sat}(\text{AX}\phi) &= Q \setminus \text{Pre}(Q \setminus \text{Sat}(\phi)) \\
\text{Sat}(\text{EF}\phi) &= \text{Pre}^*(\text{Sat}(\phi))
\end{align*}
\]
Symbolic features

- symbolic representations of the state sets
- functions to manipulate these symbolic representations
Symbolic features

- symbolic representations of the state sets
- functions to manipulate these symbolic representations

**Example**

- suppose the automaton has 2 integer variables $a, b \in \{0, \ldots, 255\}$
- each state is a triple $(q, v_a, v_b)$ where $v_a$ and $v_b$ are values for $a$ and $b$
- the set of reachable states can contain $|Q| \times 256 \times 256$ states (huge!)
- a possible symbolic representation could be $(q_2, 3, \_)$ for all states in $q_2$ with $a = 3$ and any value for $b$
Requirements for symbolic model-checking

1. symbolic representation of $\text{Sat}(p)$ for each proposition $p \in \text{Prop}$
2. algorithm to compute a symbolic representation of $\text{Pre}(S)$ from a symbolic representation of $S$
3. algorithms to compute the complement, union and intersection of symbolic representations of the sets
4. algorithm to compare symbolic representations of sets
Binary Decision Diagrams

- **data structure** commonly used for the symbolic representation of state sets
- **Efficiency**: cheap basic operations, compact data structure
- **Simplicity**: data structure and associated algorithms simple to describe and implement
- **Easy adaptation**: appropriate for problems dealing with loosely correlated data
- **Generality**: not tied to a particular family of automata
BDD structure

$n$ boolean variables $x_1, \ldots, x_n$

- suppose $n = 4$. $\langle b_1, b_2, b_3, b_4 \rangle$ associates values with $x_1, \ldots, x_4$
**BDD structure**

**n boolean variables** $x_1, \ldots, x_n$

- Suppose $n = 4$. $\langle b_1, b_2, b_3, b_4 \rangle$ associates values with $x_1, \ldots, x_4$
- Let us represent $S = \{ \langle b_1, b_2, b_3, b_4 \rangle \mid (b_1 \lor b_3) \land (b_2 \implies b_4) \}$
Symbolic model-checking
Binary Decision Diagrams

BDD structure

\( n \) boolean variables \( x_1, \ldots, x_n \)

- suppose \( n = 4 \). \( \langle b_1, b_2, b_3, b_4 \rangle \) associates values with \( x_1, \ldots, x_4 \)
- Let us represent \( S = \{ \langle b_1, b_2, b_3, b_4 \rangle \mid (b_1 \lor b_3) \land (b_2 \Rightarrow b_4) \} \)
- Possible representations:
  \[ S = \{ \langle F, F, T, F \rangle, \langle F, F, T, T \rangle, \langle F, T, T, T \rangle, \langle T, F, F, F \rangle, \langle T, F, F, T \rangle, \langle T, F, T, F \rangle, \langle T, F, T, T \rangle, \langle T, T, F, T \rangle, \langle T, T, T, T \rangle \} \]
  \( |S| = 9 \):
- the formula itself: \( (b_1 \lor b_3) \land (b_2 \Rightarrow b_4) \)
- the formula in disjunctive normal form:
  \( (b_1 \land \neg b_2) \lor (b_1 \land b_4) \lor (b_3 \land \neg b_2) \lor (b_3 \land b_4) \)
- a decision tree
Symbolic model-checking

Binary Decision Diagrams

Representation with a decision tree

\[(b_1 \lor b_3) \land (b_2 \implies b_4)\]
BDD: a reduced decision tree

- identical subtrees are shared $\leadsto$ directed acyclic graph (dag)
- internal superfluous nodes are deleted (where no choice is possible)

$$(b_1 \lor b_3) \land (b_2 \implies b_4)$$
BDD: a reduced decision tree

- identical subtrees are shared \(\sim\) directed acyclic graph (dag)
- internal superfluous nodes are deleted (where no choice is possible)

\[(b_1 \lor b_3) \land (b_2 \implies b_4)\]
BDD: a reduced decision tree

- **identical subtrees** are shared \(\leadsto\) directed acyclic graph (dag)
- **internal superfluous nodes** are deleted (where no choice is possible)

\[(b_1 \lor b_3) \land (b_2 \implies b_4)\]
BDD: a reduced decision tree

- identical subtrees are shared \( \leadsto \) directed acyclic graph (\textit{dag})
- internal superfluous nodes are deleted (where no choice is possible)

\[(b_1 \lor b_3) \land (b_2 \implies b_4)\]
**BDD: a reduced decision tree**

- identical subtrees are shared \(\rightsquigarrow\) directed acyclic graph (dag)
- internal superfluous nodes are deleted (where no choice is possible)

\[(b_1 \lor b_3) \land (b_2 \implies b_4)\]
BDD: a reduced decision tree

- **identical subtrees** are shared \( \rightsquigarrow \) directed acyclic graph (dag)
- **internal superfluous nodes** are deleted (where no choice is possible)

\[(b_1 \lor b_3) \land (b_2 \Rightarrow b_4)\]
BDD: a reduced decision tree

- identical subtrees are shared \( \leadsto \) directed acyclic graph (dag)
- internal superfluous nodes are deleted (where no choice is possible)

\[(b_1 \lor b_3) \land (b_2 \implies b_4)\]
Testing whether a tuple belongs to the set

Are $\langle T, F, T, F \rangle$, $\langle F, F, T, F \rangle$ in $S$?
Testing whether a tuple belongs to the set

Are \( \langle T, F, T, F \rangle, \langle F, F, T, F \rangle \) in \( S \)?
Testing whether a tuple belongs to the set

Are $\langle T, F, T, F \rangle$, $\langle F, F, T, F \rangle$ in $S$?
Testing whether a tuple belongs to the set $S$

Are $\langle T, F, T, F \rangle$, $\langle F, F, T, F \rangle$ in $S$?
Testing whether a tuple belongs to the set

Are \(\langle T, F, T, F \rangle, \langle F, F, T, F \rangle\) in \(S\)?
Testing whether a tuple belongs to the set

Are \( \langle T, F, T, F \rangle, \langle F, F, T, F \rangle \) in \( S \)?
Testing whether a tuple belongs to the set

Are \( \langle T, F, T, F \rangle, \langle F, F, T, F \rangle \) in \( S \)?
Testing whether a tuple belongs to the set

Are \( \langle T, F, T, F \rangle \), \( \langle F, F, T, F \rangle \) in \( S \)?
Testing whether a tuple belongs to the set

Are $\langle T, F, T, F \rangle$, $\langle F, F, T, F \rangle$ in $S$?
Exercise

BDD for \( \neg((b_1 \land (b_2 \lor b_4) \land b_5) \lor \neg b_3) \lor (b_4 \implies (b_3 \land b_5)) \)
Exercise

BDD for $\neg((b_1 \land (b_2 \lor b_4) \land b_5) \lor \neg b_3) \lor (b_4 \implies (b_3 \land b_5))$
Exercise

BDD for $\neg((b_1 \land (b_2 \lor b_4) \land b_5) \lor \neg b_3) \lor (b_4 \implies (b_3 \land b_5))$ with ordering $b_3, b_4, b_5, b_1, b_2$
Exercise

BDD for $\neg((b_1 \land (b_2 \lor b_4) \land b_5) \lor \neg b_3) \lor (b_4 \implies (b_3 \land b_5))$ with ordering $b_3, b_4, b_5, b_1, b_2$
Advantages of BDDs

- **small representations**
- existence of a **canonical** BDD structure:
  - unicity for a **fixed order** of the variables
  - test the equivalence of two symbolic representations
- test the **emptiness**
- simple operations: **complement, union, intersection, projection**
Advantages of BDDs

- small representations
- existence of a canonical BDD structure:
  - unicity for a fixed order of the variables
  - test the equivalence of two symbolic representations

Identical canonical BDDs

- test the emptyness

- simple operations: complement, union, intersection, projection
Advantages of BDDs

- small representations
- existence of a canonical BDD structure:
  - unicity for a fixed order of the variables
  - test the equivalence of two symbolic representations

Identical canonical BDDs

- test the emptiness

Reduced to the F leaf

- simple operations: complement, union, intersection, projection
Exercise

Complement
Exercise

Complement
Exercise

Union

Laure Petrucci
Advanced modelling techniques
Exercise

Union

Symbolic model-checking
Binary Decision Diagrams

Laure Petrucci

Advanced modelling techniques
Exercise

Intersection

```
Symbolic model-checking
Binary Decision Diagrams

Laure Petrucci
Advanced modelling techniques

83 / 95
```
Exercise

Intersection

Symbolic model-checking  Binary Decision Diagrams

Laure Petrucci  Advanced modelling techniques

83 / 95
Exercise

Projection $S[b_3 := T]$
Exercise

Projection $S[b_3 := T]$
Encoding of states

- boolean encoding of states
- boolean encoding of each individual variable

Let us consider an automaton with:

- \( Q = \{q_0, \ldots, q_6\} \)
- an integer variable \( \text{digit} \in \{0, \ldots, 9\} \)
- a boolean variable \( \text{ready} \)

It can be encoded with 8 bits. For example, \( \langle q_3, 8, F \rangle \) is represented by:

\[
\left( F, T, T, T, F, F, F, F \right)
\]

\[
\left( q_3, b_1^1, b_2^1, b_3^1, b_1^2, b_2^2, b_3^2, b_1^3 \right)
\]
Representing a set of states

\[ \text{Sat}(\text{ready} \implies (\text{digit} > 2)) \]
Representing a set of states

\[ \text{Sat}(\text{ready } \implies (\text{digit } > 2)) \]
Representing a transition

Transition seen as a pair of states

\[ \langle q_3, 8, F \rangle \rightarrow \langle q_5, 0, F \rangle \] is represented by:

\[
\begin{align*}
8 : (b_1^1, b_1^2, b_2^3, b_2^2, b_2^3, b_3^1, b_1^1, b_2^1, b_2^1, b_2^2, b_2^2, b_1^2, b_1^2, b_2^3, b_2^3, b_1^1)
\end{align*}
\]

\[ q_5 : (F, T, T, F, F, F, F, F, F) \]
Reachability Properties

- Reachability in temporal logic
- Computation of the reachability graph
Reachability properties

How to characterise reachability properties?

A reachability property states that some particular situation can be reached.

It may:

- be simple
- be conditional: restrict the form of paths reaching the state
- apply to any reachable state

Often, the negation of reachability is the interesting property.
Reachability properties

- we can obtain $n < 0$
- we can enter the critical section
- we cannot have $n < 0$
- we cannot reach the *crash* state
- we can enter the critical section without traversing $n = 0$
- we can always return to the initial state
- we can return to the initial state
Reachability properties

Examples

- we can obtain $n < 0$ (simple)
- we can enter the critical section
- we cannot have $n < 0$
- we cannot reach the crash state
- we can enter the critical section without traversing $n = 0$
- we can always return to the initial state
- we can return to the initial state
## Reachability properties

**Examples**

- we can obtain $n < 0$ (simple)
- we can enter the critical section (simple)
- we cannot have $n < 0$
- we cannot reach the *crash* state
- we can enter the critical section without traversing $n = 0$
- we can always return to the initial state
- we can return to the initial state
Reachability properties

Examples

- we can obtain $n < 0$ (simple)
- we can enter the critical section (simple)
- we cannot have $n < 0$ (negation)
- we cannot reach the *crash* state
- we can enter the critical section without traversing $n = 0$
- we can always return to the initial state
- we can return to the initial state
Reachability properties

Examples

- we can obtain $n < 0$ (simple)
- we can enter the critical section (simple)
- we cannot have $n < 0$ (negation)
- we cannot reach the \textit{crash} state (negation)
- we can enter the critical section without traversing $n = 0$
- we can always return to the initial state
- we can return to the initial state
Reachability properties

Examples

- we can obtain $n < 0$ (simple)
- we can enter the critical section (simple)
- we cannot have $n < 0$ (negation)
- we cannot reach the \textit{crash} state (negation)
- we can enter the critical section without traversing $n = 0$ (conditional)
- we can always return to the initial state
- we can return to the initial state
Reachability properties

Examples

- we can obtain $n < 0$ (simple)
- we can enter the critical section (simple)
- we cannot have $n < 0$ (negation)
- we cannot reach the crash state (negation)
- we can enter the critical section without traversing $n = 0$ (conditional)
- we can always return to the initial state (any reachable state)
- we can return to the initial state
Reachability properties

Examples

- we can obtain $n < 0$ (simple)
- we can enter the critical section (simple)
- we cannot have $n < 0$ (negation)
- we cannot reach the crash state (negation)
- we can enter the critical section without traversing $n = 0$ (conditional)
- we can always return to the initial state (any reachable state)
- we can return to the initial state (simple)
Reachability in temporal logic

Form of formulae in CTL

- use the \textbf{EF} combinator: $\text{EF}\phi$
- $\phi$ is a propositional formula \textit{without temporal combinators}
- \textbf{E_U} for conditional reachability
- \textit{nesting AG and EF} when considering any reachable state
Reachability in temporal logic

Examples

- we can obtain $n < 0$:
- we can enter the critical section:
- we cannot have $n < 0$:
- we cannot reach the *crash* state:
- we can enter the critical section without traversing $n = 0$:
- we can always return to the initial state:
- we can return to the initial state:
Examples

- we can obtain \( n < 0 \): \( \text{EF}(n < 0) \)
- we can enter the critical section:
- we cannot have \( n < 0 \):
- we cannot reach the \textit{crash} state:
- we can enter the critical section without traversing \( n = 0 \):
- we can always return to the initial state:
- we can return to the initial state:
Reachability in temporal logic

Examples

- we can obtain \( n < 0 \): \( \text{EF}(n < 0) \)
- we can enter the critical section: \( \text{EFcs} \)
- we cannot have \( n < 0 \):
- we cannot reach the \emph{crash} state:
- we can enter the critical section without traversing \( n = 0 \):
- we can always return to the initial state:
- we can return to the initial state:
Reachability in temporal logic

Examples

- we can obtain $n < 0$: $\text{EF}(n < 0)$
- we can enter the critical section: $\text{EF}_{cs}$
- we cannot have $n < 0$: $\neg \text{EF}(n < 0) \equiv \text{AG}(n \geq 0)$
- we cannot reach the crash state:
- we can enter the critical section without traversing $n = 0$:
- we can always return to the initial state:
- we can return to the initial state:
Reachability in temporal logic

Examples

- we can obtain $n < 0$: $\text{EF}(n < 0)$
- we can enter the critical section: $\text{EFcs}$
- we cannot have $n < 0$: $\neg\text{EF}(n < 0) \equiv \text{AG}(n \geq 0)$
- we cannot reach the crash state: $\neg\text{EFcrash} \equiv \text{AG}\neg\text{crash}$
- we can enter the critical section without traversing $n = 0$:
  - we can always return to the initial state:
  - we can return to the initial state:
Reachability in temporal logic

Examples

- we can obtain $n < 0$: $\text{EF}(n < 0)$
- we can enter the critical section: $\text{EF}cs$
- we cannot have $n < 0$: $\neg\text{EF}(n < 0) \equiv \text{AG}(n \geq 0)$
- we cannot reach the crash state: $\neg\text{EF}crash \equiv \text{AG} \neg crash$
- we can enter the critical section without traversing $n = 0$: $E(n \neq 0)Ucs$
- we can always return to the initial state:
- we can return to the initial state:
Reachability in temporal logic

Examples

- we can obtain $n < 0$: $\text{EF}(n < 0)$
- we can enter the critical section: $\text{EF}_{cs}$
- we cannot have $n < 0$: $\neg \text{EF}(n < 0) \equiv \text{AG}(n \geq 0)$
- we cannot reach the crash state: $\neg \text{EF}_{crash} \equiv \text{AG} \neg \text{crash}$
- we can enter the critical section without traversing $n = 0$: $\text{E}(n \neq 0) \cup cs$
- we can always return to the initial state: $\text{AGEF}_{\text{init}}$
- we can return to the initial state:
Reachability in temporal logic

Examples

- we can obtain $n < 0$: $\text{EF}(n < 0)$
- we can enter the critical section: $\text{EFcs}$
- we cannot have $n < 0$: $\neg\text{EF}(n < 0) \equiv \text{AG}(n \geq 0)$
- we cannot reach the crash state: $\neg\text{EFcrash} \equiv \text{AG}\neg\text{crash}$
- we can enter the critical section without traversing $n = 0$: $\text{E}(n \neq 0)\cup\text{cs}$
- we can always return to the initial state: $\text{AGEFinit}$
- we can return to the initial state: $\text{EFinit}$
Computation of the reachability graph

Forward chaining
- start from the initial state
- add its successors
- continue until saturation

**Drawback:** potential explosion of the set being constructed
Computation of the reachability graph

Backward chaining

Construct the set of states which can lead to some target states

- start from target states
- add their immediate predecessors
- continue until saturation
- test whether some initial state is in the computed set

Drawbacks:

- identify target states
- computing predecessors can be more difficult than computing successors (e.g. for automata with variables)
- target states may be unreachable
Computation of the reachability graph

On-the-fly exploration

- check the property during exploration
- only partially construct the state space