Lecture 4. Formal specifications: LTL, CTL
ELEC-E8110 Automation Systems Synthesis and Analysis

Igor Buzhinsky

igor.buzhinskii@aalto.fi

Aalto University
School of Electrical Engineering

2018
State (reachability) graph of a system

- Nodes: all reachable states of the system
- If the system is modular, then the state of the system consists of the state of all its modules
- Directed edges: one-step evolutions of the state
- Multiple outgoing edges are possible from each state, i.e. nondeterminism is common
State graph: example

NCES module (actually, a Petri net)
NCES module (actually, a Petri net)  
State graph, state = $p_1p_2p_3$
Kripke structures

- Formalization of a state graph

Let $\text{AP}$ be the a finite set of so-called atomic propositions

Then $M = (S, I, T, L)$ is a Kripke structure, where:

- $S$ is a finite set of states
- $I \subset S$ is a set of initial states
- $T \subset S \times S$ is a transition relation
- $L : S \to 2^{\text{AP}}$ is a labeling function

No deadlock assumption: $\forall s \in S \exists s' \in S : (s, s') \in T$
State graph interpreted as a Kripke structure

AP = \{ \]
| p_i = j |
| i = 1..3, j = 0..2 |

S: nodes of this graph
I \subset S = \{ 101 \}
T \subset S \times S: edges of the graph, e.g. (002, 011), (011, 002), ...
L: S \rightarrow 2^{AP}: token assignments (markings) in each state, e.g. 
L(020) = \{ p_0 = 0, p_1 = 2, p_3 = 0 \}

Specifications can be interpreted as predicates over Kripke structures
State graph interpreted as a Kripke structure

\[ AP = \{ \text{a} = 0 \} \]

- \( I \subset S = \{ 101 \} \)
- \( T \subset S \times S \): edges of the graph, e.g. (002, 011), (011, 002), ...
- \( L : S \rightarrow 2^{AP} \): token assignments (markings) in each state, e.g. \( L(020) = \{ p_0 = 0, p_1 = 2, p_3 = 0 \} \)

Specifications can be interpreted as predicates over Kripke structures.
State graph interpreted as a Kripke structure

\[ AP = \{ "p_i = j" \mid i = 1..3, j = 0..2 \} \]

\[ S: \]
State graph interpreted as a Kripke structure

- $\text{AP} = \{ "p_i = j" \mid i = 1..3, j = 0..2 \}$
- $S$: nodes of this graph
- $I \subseteq S =$
State graph interpreted as a Kripke structure

- **AP** = \{ “p_i = j” | i = 1..3, j = 0..2 \}
- **S**: nodes of this graph
- **I ⊂ S** = \{101\}
- **Note**: in UPPAAL and NCES models there is always one initial state!
- **T ⊂ S × S**: 

![State Graph Diagram]

Specifications can be interpreted as predicates over Kripke structures
State graph interpreted as a Kripke structure

- $AP = \{ "p_i = j" | i = 1..3, j = 0..2 \}$
- $S$: nodes of this graph
- $I \subset S = \{101\}$
- Note: in UPPAAL and NCES models there is always one initial state!
- $T \subset S \times S$: edges of the graph, e.g. $(002, 011), (011, 002), \ldots$
- $L : S \rightarrow 2^{AP}$:
State graph interpreted as a Kripke structure

- \( AP = \{ "p_i = j" \mid i = 1..3, j = 0..2 \} \)
- \( S \): nodes of this graph
- \( I \subseteq S = \{ 101 \} \)
- Note: in UPPAAL and NCES models there is always one initial state!
- \( T \subseteq S \times S \): edges of the graph, e.g. \((002, 011), (011, 002), ...\)
- \( L : S \rightarrow 2^{AP} \): token assignments (markings) in each state, e.g. 
  \( L(020) = \{ "p_0 = 0", "p_1 = 2", "p_3 = 0" \} \)
State graph interpreted as a Kripke structure

- $\text{AP} = \{ "p_i = j" \mid i = 1..3, j = 0..2 \}$
- $S$: nodes of this graph
- $I \subset S = \{101\}$
- Note: in UPPAAL and NCES models there is always one initial state!
- $T \subset S \times S$: edges of the graph, e.g. $(002, 011), (011, 002), ...$
- $L : S \rightarrow 2^{\text{AP}}$: token assignments (markings) in each state, e.g. $L(020) = \{ "p_0 = 0", "p_1 = 2", "p_3 = 0" \}$

**Specifications** can be interpreted as predicates over Kripke structures
Kripke structures / state graphs for UPPAAL models?

plant

controller

Note: we ignore timed capabilities of UPPAAL by now
Kripke structures / state graphs for UPPAAL models?

- **AP**: whether a state machine is in a certain state, whether a variable has a certain value
- **S**:

\[
S = (s_1, \ldots, s_k, v_1, \ldots, v_m) \in S \text{ if it is a reachable combination of states and variable values}
\]

- **I**: \( \subset S \): single initial state
- **T**: \( \subset S \times S \): valid state transitions
- **L**: \( S \to 2^{AP} \): individual states and variable values

Note: we ignore timed capabilities of UPPAAL by now.
Kripke structures / state graphs for UPPAAL models?

- **AP**: whether a state machine is in a certain state, whether a variable has a certain value
- **S**: \((s_1, ..., s_k, v_1, ..., v_m) \in S\) if it is a reachable combination of states and variable values
- **I \subset S**: 

---

**plant**

- **s_1**: extend & turn == 1, turn = 0
- **s_2**: extend & turn == 1, retract & turn == 1, retract = true, turn = 0
- **s_3**: extend & turn == 1, retract & turn == 1, retract = true, turn = 0
- **s_4**: retract & turn == 1, extended = false, turn = 0

**controller**

- **s_start**: extend = true, retract = false
- **s_go**: extended & turn == 0, turn = 1
- **s_return**: extended & turn == 0, extend = false, retract = true, turn = 1
- retract & turn == 0, extend = true, retract = false, turn = 1
AP: whether a state machine is in a certain state, whether a variable has a certain value

$S$: $(s_1, ..., s_k, v_1, ..., v_m) \in S$ if it is a reachable combination of states and variable values

$I \subset S$: single initial state $(s_{01}, ..., s_{0k}, v_{01}, ..., v_{0m})$ composed of initial individual states and variable values

$T \subset S \times S$: 

**plant**

1. $s_1$
   - extend & turn == 1
   - retract & turn == 1
   - retract = false, turn = 0
   - retract = true, turn = 0

2. $s_2$
   - extend & retract & turn == 1
   - turn = 0

3. $s_3$
   - extend & turn == 1
   - retract & turn == 1
   - extended = true, turn = 0
   - extended = false, turn = 0

4. $s_4$
   - retract & turn == 1
   - turn = 0

**controller**

1. $s_{start}$
   - turn == 0
   - extend = true, retract = false

2. $s_{go}$
   - extended & turn == 0
   - turn = 1
   - extend = false, retract = true, turn = 1
   - retract & turn == 0
   - extend = true, retract = false, turn = 1

3. $s_{return}$
   - extended & turn == 0
   - turn = 1
   - extend = false, retract = true, turn = 1
Kripke structures / state graphs for UPPAAL models?

- **AP**: whether a state machine is in a certain state, whether a variable has a certain value
- **$S$**: $(s_1, ..., s_k, v_1, ..., v_m) \in S$ if it is a reachable combination of states and variable values
- **$I \subset S$**: single initial state $(s_{01}, ..., s_{0k}, v_{01}, ..., v_{0m})$ composed of initial individual states and variable values
- **$T \subset S \times S$**: valid state transitions
- **$L : S \rightarrow 2^{\text{AP}}$**:
Kripke structures / state graphs for UPPAAL models?

- AP: whether a state machine is in a certain state, whether a variable has a certain value
- $S$: $(s_1, ..., s_k, v_1, ..., v_m) \in S$ if it is a reachable combination of states and variable values
- $I \subset S$: single initial state $(s_{01}, ..., s_{0k}, v_{01}, ..., v_{0m})$ composed of initial individual states and variable values
- $T \subset S \times S$: valid state transitions
- $L: S \to 2^{AP}$: individual states and variable values

Note: we ignore timed capabilities of UPPAAL by now
System behaviors are paths in Kripke structures

Infinite paths are common in formal verification. This is the reason why deadlocks are undesirable.
System behaviors are paths in Kripke structures

Infinite paths are common in formal verification. This is the reason why deadlocks are undesirable.

What happens in terms of the original system?
Assume that now we have only two atomic propositions: $p$ and $q$.

All possible behaviors are infinite sequences over $2\{p,q\}$.

Example: $\{p, q\}, \{p\}, \{\}, \text{cycle}(\{q\}, \{p, q\})$.
Single behavior view

- Assume that now we have only two atomic propositions: $p$ and $q$
- All possible behaviors are infinite sequences over $2\{p,q\}$
- Example: $\{p, q\}, \{p\}, \{\}, \text{cycle}(\{q\}, \{p, q\})$
- Boolean logic is able to characterize single elements of such sequences
- Can we somehow introduce predicates over infinite sequences of atomic propositions?
- For example, to formulate a specification: each $p$ is followed by $\neg p$ on the next step (which is false for the example)
Linear temporal logic (LTL)

- Formal language which extends the usual propositional Boolean logic
- Variables: atomic propositions, e.g. $p$ and $q$
- Usual Boolean operators are allowed, e.g. $p \rightarrow q$ (i.e. $\neg p \lor q$) is an LTL formula, but it refers to the **first element** of an infinite sequence
Linear temporal logic (LTL)

- Formal language which extends the usual propositional Boolean logic
- Variables: atomic propositions, e.g. $p$ and $q$
- Usual Boolean operators are allowed, e.g. $p \rightarrow q$ (i.e. $\neg p \lor q$) is an LTL formula, but it refers to the first element of an infinite sequence

**Temporal operators**

- $G$: globally (always), e.g. $G(p \rightarrow q)$ means "in each element of the sequence, $p \rightarrow q$ holds"
- $F$: in the future, e.g. $F(p \rightarrow q)$ means "for some element of the sequence, $p \rightarrow q$ holds"
- $X$: on the next step, e.g. $X(p \rightarrow q)$ means "$p \rightarrow q$ holds for the second element of the sequence"
- $U$: until (binary operator), e.g. $p U q$ means "$q$ must happen at some step, and the sequence must satisfy $p$ until (non-inclusive) $q$ happens"
Linear temporal logic (LTL)

- Formal language which extends the usual propositional Boolean logic
- Variables: atomic propositions, e.g. $p$ and $q$
- Usual Boolean operators are allowed, e.g. $p \rightarrow q$ (i.e. $\neg p \lor q$) is an LTL formula, but it refers to the **first element** of an infinite sequence

**Temporal operators**

- $G$: globally (always), e.g. $G(p \rightarrow q)$ means “in each element of the sequence, $p \rightarrow q$ holds”
Linear temporal logic (LTL)

- Formal language which extends the usual propositional Boolean logic
- Variables: atomic propositions, e.g. \( p \) and \( q \)
- Usual Boolean operators are allowed, e.g. \( p \rightarrow q \) (i.e. \( \neg p \lor q \)) is an LTL formula, but it refers to the first element of an infinite sequence

Temporal operators

- \( G \): globally (always), e.g. \( G(p \rightarrow q) \) means “in each element of the sequence, \( p \rightarrow q \) holds”
- \( F \): in the future, e.g. \( F(p \rightarrow q) \) means “for some element of the sequence, \( p \rightarrow q \) holds”
Linear temporal logic (LTL)

- Formal language which extends the usual propositional Boolean logic
- Variables: atomic propositions, e.g. $p$ and $q$
- Usual Boolean operators are allowed, e.g. $p \rightarrow q$ (i.e. $\neg p \lor q$) is an LTL formula, but it refers to the first element of an infinite sequence

Temporal operators
- $G$: globally (always), e.g. $G(p \rightarrow q)$ means “in each element of the sequence, $p \rightarrow q$ holds”
- $F$: in the future, e.g. $F(p \rightarrow q)$ means “for some element of the sequence, $p \rightarrow q$ holds”
- $X$: on the next step, e.g. $X(p \rightarrow q)$ means “$p \rightarrow q$ holds for the second element of the sequence”
Linear temporal logic (LTL)

- Formal language which extends the usual propositional Boolean logic
- Variables: atomic propositions, e.g. $p$ and $q$
- Usual Boolean operators are allowed, e.g. $p \rightarrow q$ (i.e. $\neg p \lor q$) is an LTL formula, but it refers to the **first element** of an infinite sequence

**Temporal operators**

- **G**: globally (always), e.g. $G(p \rightarrow q)$ means “in each element of the sequence, $p \rightarrow q$ holds”
- **F**: in the future, e.g. $F(p \rightarrow q)$ means “for some element of the sequence, $p \rightarrow q$ holds”
- **X**: on the next step, e.g. $X(p \rightarrow q)$ means “$p \rightarrow q$ holds for the second element of the sequence”
- **U**: until (binary operator), e.g. $p U q$ means “$q$ must happen at some step, and the sequence must satisfy $p$ until (non-inclusive) $q$ happens”
Examples of LTL formulae

- Path 1: \( \{p, q\}, \{p\}, \{\}, \text{cycle}(\{q\}, \{p, q\}) \)
- Path 2: \( \text{cycle}(\{p, q\}) \)
- Path 3: \( \{\}, \text{cycle}(\{p\}, \{p, q\}, \{q\}) \)

- \( f_1 = G\ p \)
Examples of LTL formulae

- Path 1: \{p, q\}, \{p\}, \{}, \text{cycle}(\{q\}, \{p, q\})
- Path 2: \text{cycle}(\{p, q\})
- Path 3: \{}, \text{cycle}(\{p\}, \{p, q\}, \{q\})

- $f_1 = G\ p$ – path 2
- $f_2 = F(\neg p \land \neg q)$
Examples of LTL formulae

Path 1: \{p, q\}, \{p\}, {}, cycle({q}, {p, q})
Path 2: cycle({p, q})
Path 3: {}, cycle({p}, {p, q}, {q})

\( f_1 = \mathbf{G} p \) – path 2
\( f_2 = \mathbf{F}(\neg p \land \neg q) \) – paths 1, 3
\( f_3 = p \mathbf{U}(\neg p \land \neg q) \)
Examples of LTL formulae

- Path 1: \{p, q\}, \{p\}, \{\}, cycle(\{q\}, \{p, q\})
- Path 2: cycle(\{p, q\})
- Path 3: \{\}, cycle(\{p\}, \{p, q\}, \{q\})

- \(f_1 = G \ p\) – path 2
- \(f_2 = F(\neg p \land \neg q)\) – paths 1, 3
- \(f_3 = p \ U(\neg p \land \neg q)\) – paths 1, 3

Temporal operators can be applied to arbitrary LTL formulae!
- \(f_4 = X X X X \ p\)
Examples of LTL formulae

- Path 1: $\{p, q\}, \{p\}, \{\}, \text{cycle}(\{q\}, \{p, q\})$
- Path 2: $\text{cycle}(\{p, q\})$
- Path 3: $\{\}, \text{cycle}(\{p\}, \{p, q\}, \{q\})$

- $f_1 = \mathbf{G} p$ – path 2
- $f_2 = \mathbf{F}(\neg p \land \neg q)$ – paths 1, 3
- $f_3 = p \mathbf{U}(\neg p \land \neg q)$ – paths 1, 3

Temporal operators can be applied to arbitrary LTL formulae!
- $f_4 = \mathbf{X} \mathbf{X} \mathbf{X} \mathbf{X} \mathbf{X} p$ ("on the fifth step") – paths 1, 2, 3
- $f_5 = \mathbf{F} \mathbf{G}(p \land q)$
Examples of LTL formulae

- Path 1: \( \{p, q\}, \{p\}, \{\}, \text{cycle}(\{q\}, \{p, q\}) \)
- Path 2: \( \text{cycle}(\{p, q\}) \)
- Path 3: \( \{\}, \text{cycle}(\{p\}, \{p, q\}, \{q\}) \)

- \( f_1 = Gp \) – path 2
- \( f_2 = F(\neg p \land \neg q) \) – paths 1, 3
- \( f_3 = p U(\neg p \land \neg q) \) – paths 1, 3

Temporal operators can be applied to arbitrary LTL formulae!

- \( f_4 = X X X X p \) ("on the fifth step") – paths 1, 2, 3
- \( f_5 = GF(p \land q) \) ("globally from some point") – path 2
- \( f_6 = GF(p \land q) \)
Examples of LTL formulae

- Path 1: \( \{p, q\}, \{p\}, \emptyset, \text{cycle}(\{q\}, \{p, q\}) \)
- Path 2: \( \text{cycle}(\{p, q\}) \)
- Path 3: \( \emptyset, \text{cycle}(\{p\}, \{p, q\}, \{q\}) \)

\[
f_1 = G\ p \quad \text{– path 2}
f_2 = F(\neg p \land \neg q) \quad \text{– paths 1, 3}
f_3 = p \mathbf{U}(\neg p \land \neg q) \quad \text{– paths 1, 3}
\]

Temporal operators can be applied to arbitrary LTL formulae!

\[
f_4 = \mathbf{X} \mathbf{X} \mathbf{X} \mathbf{X} p \quad \text{ (“on the fifth step”) – paths 1, 2, 3}
\]

\[
f_5 = F \ G(p \land q) \quad \text{ (“globally from some point”) – path 2}
\]

\[
f_6 = G \ F(p \land q) \quad \text{ (“infinitely often”) – paths 1, 2, 3}
\]

\[
f_7 = G(p \rightarrow \mathbf{X} \ q)
\]
Examples of LTL formulae

- Path 1: \{p, q\}, \{p\}, {}, cycle(\{q\}, \{p, q\})
- Path 2: cycle(\{p, q\})
- Path 3: {}, cycle(\{p\}, \{p, q\}, \{q\})

- \(f_1 = \mathbf{G} p\) – path 2
- \(f_2 = \mathbf{F}(\neg p \land \neg q)\) – paths 1, 3
- \(f_3 = p \mathbf{U}(\neg p \land \neg q)\) – paths 1, 3

Temporal operators can be applied to arbitrary LTL formulae!

- \(f_4 = \mathbf{X} \mathbf{X} \mathbf{X} \mathbf{X} p\) (“on the fifth step”) – paths 1, 2, 3
- \(f_5 = \mathbf{F} \mathbf{G}(p \land q)\) (“globally from some point”) – path 2
- \(f_6 = \mathbf{G} \mathbf{F}(p \land q)\) (“infinitely often”) – paths 1, 2, 3
- \(f_7 = \mathbf{G}(p \rightarrow \mathbf{X} q)\) (“\(p\) is always followed by \(q\)”) – paths 2, 3
Path visualization tool

- Download from https://github.com/igor-buzhinsky/nusmv_counterexample_visualizer
- The tool supports interpreting counterexamples produced by NuSMV (will be covered later in the course), but can also be used to **structurally explain** LTL formula values on user-specified paths
- **Important atomic propositions** are highlighted

Variable view: suppose that the state of the formal model is composed on a number of variables, either Boolean or integer
- Boolean variables can be interpreted as atomic propositions right away
- Statements over integer variables (e.g. comparisons like $x > 5$) can also be interpreted as atomic propositions
Path visualization tool: example

- Visualize $G(p \rightarrow X q)$ on path $\{p, q\}, \{p\}, \emptyset, \text{cycle(}\{q\}, \{p, q\})$:

<table>
<thead>
<tr>
<th>#</th>
<th>Value</th>
<th>LTL specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>TRUE</td>
<td>$G F (p &amp; q)$</td>
</tr>
<tr>
<td>8</td>
<td>TRUE</td>
<td>$G F (p &amp; q)$</td>
</tr>
<tr>
<td>9</td>
<td>TRUE</td>
<td>$G F (p &amp; q)$</td>
</tr>
<tr>
<td>10</td>
<td>FALSE</td>
<td>$G (p \rightarrow X q)$</td>
</tr>
<tr>
<td>11</td>
<td>TRUE</td>
<td>$G (p \rightarrow X q)$</td>
</tr>
<tr>
<td>12</td>
<td>TRUE</td>
<td>$G (p \rightarrow X q)$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step</th>
<th>p</th>
<th>q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>1</td>
<td>TRUE</td>
<td>FALSE</td>
</tr>
<tr>
<td>2</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
<tr>
<td>3</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>4</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

---

Step 0 prefix

Step 1 prefix

Step 2 prefix

- Step 3 loop

- Step 4 loop
Path visualization tool: input format

-- specification G (p -> X q)
-> new state <-
  p = TRUE
  q = TRUE
-> new state <-
  p = TRUE
  q = FALSE
-> new state <-
  p = FALSE
  q = FALSE
-- Loop starts here
-> new state <-
  p = FALSE
  q = TRUE
-> new state <-
  p = TRUE
  q = TRUE
LTL: simplification and equivalence rules

- $G G f = G f$
- $F F f = F f$
LTL: simplification and equivalence rules

- $G G f = G f$
- $F F f = F f$
- $G X f = X G f$
- $F X f = X F f$
LTL: simplification and equivalence rules

- $\mathit{G} \mathit{G} f = \mathit{G} f$
- $\mathit{F} \mathit{F} f = \mathit{F} f$
- $\mathit{G} \mathit{X} f = \mathit{X} \mathit{G} f$
- $\mathit{F} \mathit{X} f = \mathit{X} \mathit{F} f$
- $\neg \mathit{G}(f) = \mathit{F}(\neg f)$
- $\neg \mathit{F}(f) = \mathit{G}(\neg f)$
LTL verification: definition

- Kripke structure $M$ satisfies LTL formula $f$ (written: $M \models f$), if all paths in $M$ which start in $M$'s initial states satisfy $f$.

Quiz: which of these LTL formulae are satisfied by the KS on the right? Why?

- $f_1 = G p$
- $f_2 = F (\neg p \land \neg q)$
- $f_3 = p U (\neg p \land \neg q)$
- $f_4 = XX XX XX p$
- $f_5 = F G (p \land q)$
- $f_6 = G F (p \land q)$
- $f_7 = G (p \rightarrow X q)$
Kripke structure $M$ satisfies LTL formula $f$ (written: $M \models f$), if all paths in $M$ which start in $M$’s initial states satisfy $f$

**Quiz:** which of these LTL formulae are satisfied by the KS on the right? Why?

- $f_1 = G p$
- $f_2 = F(\neg p \land \neg q)$
- $f_3 = p \mathbf{U}(\neg p \land \neg q)$
- $f_4 = \mathbf{XXXX} p$
- $f_5 = F \mathbf{G}(p \land q)$
- $f_6 = \mathbf{G} F(p \land q)$
- $f_7 = G(p \rightarrow \mathbf{X} q)$
Only $f_6 = \mathbf{G} \mathbf{F} (p \land q)$
We wish to check whether $f$ holds for Kripke structure $M$

Automata-theoretic approach

$\neg f$ is converted to a so-called \textbf{B"uchi automaton}, which is an acceptor over infinite words which satisfy $\neg f$

$M$ is composed with this automaton

If the composition accepts at least one infinite word, then this word satisfies $\neg f$ and belongs to $M$, so $f$ is false, and the obtained word is a \textbf{counterexample}

Otherwise, $f$ is true

We won’t go into details
Two cylinders system

- The extension of each cylinder is discretized into four intervals.
- When both cylinders share interval 4, they collide.
- A workpiece can be placed into the shared interval.
- If a cylinder reaches interval 4 and there is a workpiece, it is pushed.
Atomic propositions: $h_1, h_2, h_3, h_4$ (displacements of the horizontal cylinder), $v_1, v_2, v_3, v_4$ (displacements of the vertical cylinder), $w$ (workpiece is present)

Cylinder has a position:
Atomic propositions: $h_1, h_2, h_3, h_4$ (displacements of the horizontal cylinder), $v_1, v_2, v_3, v_4$ (displacements of the vertical cylinder), $w$ (workpiece is present)

Cylinder has a position: $\mathbf{G}(h_1 \lor h_2 \lor h_3 \lor h_4), \mathbf{G}(v_1 \lor v_2 \lor v_3 \lor v_4)$

Cylinder can’t have more than one position:
Atomic propositions: $h_1$, $h_2$, $h_3$, $h_4$ (displacements of the horizontal cylinder), $v_1$, $v_2$, $v_3$, $v_4$ (displacements of the vertical cylinder), $w$ (workpiece is present)

Cylinder has a position: $\mathbf{G}(h_1 \lor h_2 \lor h_3 \lor h_4)$, $\mathbf{G}(v_1 \lor v_2 \lor v_3 \lor v_4)$

Cylinder can’t have more than one position:
$\mathbf{G}(\neg(h_1 \land h_2) \land \neg(h_1 \land h_3) \land \neg(h_1 \land h_4) \land \neg(h_2 \land h_3) \land \neg(h_2 \land h_4) \land \neg(h_3 \land h_4))$,
$\mathbf{G}(\neg(v_1 \land v_2) \land \neg(v_1 \land v_3) \land \neg(v_1 \land v_4) \land \neg(v_2 \land v_3) \land \neg(v_2 \land v_4) \land \neg(v_3 \land v_4))$

If a cylinder is fully extended, then there is no workpiece:
LTL specification for the two cylinders system: plant model

- Atomic propositions: \( h_1, h_2, h_3, h_4 \) (displacements of the horizontal cylinder), \( v_1, v_2, v_3, v_4 \) (displacements of the vertical cylinder), \( w \) (workpiece is present)

- Cylinder has a position: \( G(h_1 \lor h_2 \lor h_3 \lor h_4), G(v_1 \lor v_2 \lor v_3 \lor v_4) \)

- Cylinder can’t have more than one position:
  \[
  G(\neg(h_1 \land h_2) \land \neg(h_1 \land h_3) \land \neg(h_1 \land h_4) \land \neg(h_2 \land h_3) \land \neg(h_2 \land h_4) \land \neg(h_3 \land h_4)),
  
  G(\neg(v_1 \land v_2) \land \neg(v_1 \land v_3) \land \neg(v_1 \land v_4) \land \neg(v_2 \land v_3) \land \neg(v_2 \land v_4) \land \neg(v_3 \land v_4))
  
- If a cylinder is fully extended, then there is no workpiece:
  \[
  G(h_4 \lor v_4 \rightarrow \neg w)
  
  ...

- Such specifications can help “debug” the plant model
LTL specification for the two cylinders system: requirements for the controller

- We won’t use any additional atomic propositions: all we need can be specified in terms of the plant!
LTL specification for the two cylinders system: requirements for the controller

- We won’t use any additional atomic propositions: all we need can be specified in terms of the plant!

- Cylinders do not collide:
LTL specification for the two cylinders system: requirements for the controller

- We won’t use any additional atomic propositions: all we need can be specified in terms of the plant!

- Cylinders do not collide: $\neg (h_4 \land v_4)$
LTL specification for the two cylinders system: requirements for the controller

- We won’t use any additional atomic propositions: all we need can be specified in terms of the plant!
- Cylinders do not collide: $\mathbf{G} \neg (h_4 \land v_4)$
- When a workpiece appears, it must be eventually pushed away:
LTL specification for the two cylinders system: requirements for the controller

- We won’t use any additional atomic propositions: all we need can be specified in terms of the plant!

- Cylinders do not collide: $\mathbf{G} \neg (h_4 \land v_4)$

- When a workpiece appears, it must be eventually pushed away: $\mathbf{G}(w \rightarrow \mathbf{F} \neg w)$
LTL specification for the two cylinders system: requirements for the controller

- We won’t use any additional atomic propositions: all we need can be specified in terms of the plant!
- Cylinders do not collide: $G\neg(h_4 \land v_4)$
- When a workpiece appears, it must be eventually pushed away: $G(w \rightarrow F\neg w)$
- Cylinders iterate (each new workpiece is pushed by a different cylinder):
LTL specification for the two cylinders system: requirements for the controller

- We won’t use any additional atomic propositions: all we need can be specified in terms of the plant!

- Cylinders do not collide: $\mathbf{G} \neg (h_4 \land v_4)$

- When a workpiece appears, it must be eventually pushed away: $\mathbf{G} (w \rightarrow \mathbf{F} \neg w)$

- Cylinders iterate (each new workpiece is pushed by a different cylinder):
  $\mathbf{G}((h_4 \land (\mathbf{X} \neg h_4) \land \mathbf{F} w) \rightarrow \mathbf{X}((\neg w \land \neg v_4 \land \neg h_4) \mathbf{U}(w \land (w \mathbf{U}(v_4 \land \neg h_4))))))$, and the same for the other cylinder
In LTL, there is always an implicit quantification over all paths starting in initial states.

In **CTL**, all temporal operators are annotated with quantifiers.

CTL formulae characterize not infinite sequences, but rather states of the Kripke structure.

A Kripke structure satisfies a CTL formula, if **all its initial states** satisfy this formula.

Let $s$ be a state of the KS, then $s \models f$ means $s$ satisfies $f$. 
$s \models \textbf{EX}(f)$ ("exists next", not supported by UPPAAL): there is a successor of $s$ where $f$ holds
$s \models \textbf{AX}(f)$ ("for all next", not supported by UPPAAL): in all successors of $s$, $f$ holds
CTL: temporal operator $\text{EF}$

$s \models \text{EF}(f)$ (“exists in the future”, $E<>$ in UPPAAL): there exists a path starting in $s$ such that $f$ becomes valid at some point of this path.
CTL: temporal operator $\text{AF}$

- $s \models \text{AF}(f)$ (“for all in the future”, $A<>$ in UPPAAL): for all possible paths starting in $s$, $f$ becomes true at some point
CTL: temporal operator $\text{EG}$

- $s \models \text{EG}(f)$ ("exists globally", $E[]$ in UPPAAL): there exists a path starting in $s$ such that $f$ holds at every state along this path.
CTL: temporal operator $\text{AG}$

- $s \models \text{AG}(f)$ (“for all globally”, $\text{A}[]$ in UPPAAL): for all possible paths starting in $s$, $f$ is always true
**CTL: temporal operators EU and AU**

- \( s \models f \text{EU} g \) ("exists until", not supported by UPPAAL): there exists a path starting in \( s \) such that \( f \) holds until (non-inclusive) \( g \), and \( g \) eventually happens.

- \( s \models f \text{AU} g \) ("for all until", not supported by UPPAAL): for all possible paths starting in \( s \), \( f \) holds until (non-inclusive) \( g \), and \( g \) eventually happens.
Are these formulae syntactically correct in LTL or CTL?

- $p \land \neg q$
Are these formulae syntactically correct in LTL or CTL?

- $p \land \neg q$ – both LTL and CTL
Are these formulae syntactically correct in LTL or CTL?

- $p \land \neg q$ – both LTL and CTL
- $\mathbf{AX}(p \rightarrow F q)$
Are these formulae syntactically correct in LTL or CTL?

- $p \land \neg q$ – both LTL and CTL
- $AX(p \rightarrow F q)$ – incorrect
Are these formulae syntactically correct in LTL or CTL?

- $p \land \neg q$ – both LTL and CTL
- $AX(p \rightarrow F q)$ – incorrect
- $FX AG q$
Are these formulae syntactically correct in LTL or CTL?

- $p \land \neg q$ – both LTL and CTL
- $AX(p \rightarrow F q)$ – incorrect
- $F X AG q$ – incorrect
Are these formulae syntactically correct in LTL or CTL?

- $p \land \neg q$ – both LTL and CTL
- $\text{AX}(p \rightarrow F q)$ – incorrect
- $F X AG q$ – incorrect
- $EX AG q$
Are these formulae syntactically correct in LTL or CTL?

- \( p \land \neg q \) – both LTL and CTL
- \( \text{AX}(p \rightarrow \text{F} q) \) – incorrect
- \( \text{F} \text{X} \text{AG} q \) – incorrect
- \( \text{EX} \text{AG} q \) – CTL
Are these formulae syntactically correct in LTL or CTL?

- $p \land \neg q$ – both LTL and CTL
- $AX(p \rightarrow F q)$ – incorrect
- $F X AG q$ – incorrect
- $EX AG q$ – CTL
- $EX \neg AG q$
Are these formulae syntactically correct in LTL or CTL?

- $p \land \neg q$ – both LTL and CTL
- $AX(p \rightarrow F q)$ – incorrect
- $FXAGq$ – incorrect
- $EXAGq$ – CTL
- $EX\neg AGq$ – CTL
Are these formulae syntactically correct in LTL or CTL?

- $p \land \neg q$ – both LTL and CTL
- $AX(p \rightarrow F q)$ – incorrect
- $F X AG q$ – incorrect
- $EX AG q$ – CTL
- $EX \neg AG q$ – CTL
- $G(p \rightarrow X X F q)$
Are these formulae syntactically correct in LTL or CTL?

- $p \land \neg q$ – both LTL and CTL
- $AX(p \rightarrow F q)$ – incorrect
- $F X AG q$ – incorrect
- $EX AG q$ – CTL
- $EX \neg AG q$ – CTL
- $G(p \rightarrow X X F q)$ – LTL
Are these formulae syntactically correct in LTL or CTL?

- \( p \land \neg q \) – both LTL and CTL
- \( AX(p \rightarrow F q) \) – incorrect
- \( F X AG q \) – incorrect
- \( EX AG q \) – CTL
- \( EX \neg AG q \) – CTL
- \( G(p \rightarrow X X F q) \) – LTL
- \( (AX p) U(EF \neg p) \)
Are these formulae syntactically correct in LTL or CTL?

- $p \land \neg q$ – both LTL and CTL
- $AX(p \rightarrow F q)$ – incorrect
- $F X AG q$ – incorrect
- $EX AG q$ – CTL
- $EX \neg AG q$ – CTL
- $G(p \rightarrow X X F q)$ – LTL
- $(AX p) U(EF \neg p)$ – incorrect
KS satisfies the CTL formula iff all its initial states satisfy it

\[
\begin{align*}
\text{f}_1 &= \mathit{AG}p \\
\text{f}_2 &= \mathit{AG}(p \lor q) \\
\text{f}_3 &= \mathit{AF}(p \land q) \\
\text{f}_4 &= \mathit{EF}(\neg p \land \neg q) \\
\text{f}_5 &= \mathit{AX} \mathit{AX} \mathit{AX} \mathit{AX}p \\
\text{f}_6 &= \mathit{EF} \mathit{EG}(p \land q) \\
\text{f}_7 &= \mathit{EG} \mathit{EF}(p \land q) \\
\text{f}_8 &= \mathit{AG}(p \rightarrow \mathit{AX}q)
\end{align*}
\]

Answer: \text{f}_2, \text{f}_3, \text{f}_6, \text{f}_7
KS satisfies the CTL formula iff all its initial states satisfy it.

Which of these CTL formulae are satisfied by the KS on the right? Why?

- $f_1 = \text{AG } p$
- $f_2 = \text{AG}(p \lor q)$
- $f_3 = \text{AF}(p \land q)$
- $f_4 = \text{EF}(\neg p \land \neg q)$
- $f_5 = \text{AX AX AX AX AX } p$
- $f_6 = \text{EF EG}(p \land q)$
- $f_7 = \text{EG EF}(p \land q)$
- $f_8 = \text{AG}(p \rightarrow \text{AX } q)$
KS satisfies the CTL formula iff all its initial states satisfy it

Which of these CTL formulae are satisfied by the KS on the right? Why?

- $f_1 = AG\ p$
- $f_2 = AG(p \lor q)$
- $f_3 = AF(p \land q)$
- $f_4 = EF(\neg p \land \neg q)$
- $f_5 = AX\ AX\ AX\ AX\ AX\ p$
- $f_6 = EF\ EG(p \land q)$
- $f_7 = EG\ EF(p \land q)$
- $f_8 = AG(p \rightarrow AX\ q)$

Answer: $f_2$, $f_3$, $f_6$, $f_7$
KS satisfies the CTL formula iff all its initial states satisfy it.

Which of these CTL formulae are satisfied by the KS on the right? Why?

- $f_1 = \text{AG } p$
- $f_2 = \text{AG}(p \lor q)$
- $f_3 = \text{AF}(p \land q)$
- $f_4 = \text{EF}(\neg p \land \neg q)$
- $f_5 = \text{AX AX AX AX AX } p$
- $f_6 = \text{EF EG}(p \land q)$
- $f_7 = \text{EG EF}(p \land q)$
- $f_8 = \text{AG}(p \rightarrow \text{AX } q)$

Answer: $f_2, f_3, f_6, f_7$
Graph theory approach

- If $f$ is just a Boolean formula, then it is trivial to check whether it holds in a desired state
- $\mathbf{AX} f$ holds in $s$, if we previously found that $f$ holds for each in all its successors
- Similar (sometimes more complex) ideas for other operators
- We won’t go into details

Symbolic approach

- Implicit representation of states via Boolean formulae
- Allows partial mitigation of the state explosion problem
- The symbolic approach will be examined in more detail later in the course
CTL verification: two cylinders

- Atomic propositions:
  \( h_1 \ldots h_4, v_1 \ldots v_4, w \)

**Quiz:** specify the following properties in CTL:

- If a cylinder is fully extended, then there is no workpiece
- Cylinders do not collide
- When a workpiece appears, it must be eventually pushed away
- Cylinders iterate
Quiz answers

If a cylinder is fully extended, then there is no workpiece:
\[ \text{AG} (h_4 \lor v_4 \rightarrow \neg w) \]

Cylinders do not collide:
\[ \text{AG} \neg (h_4 \land v_4) \]

When a workpiece appears, it must be eventually pushed away:
\[ \text{AG} (w \rightarrow \text{AF} \neg w) \]

Cylinders iterate: ???
Quiz answers

- If a cylinder is fully extended, then there is no workpiece:
  \[ \text{AG}(h_4 \lor v_4 \rightarrow \neg w) \]
- Cylinders do not collide: \[ \text{AG} \neg(h_4 \land v_4) \]
- When a workpiece appears, it must be eventually pushed away:
  \[ \text{AG}(w \rightarrow \text{AF} \neg w) \]
- Cylinders iterate: ???
### Common specification types ("patterns")

<table>
<thead>
<tr>
<th>Name</th>
<th>LTL</th>
<th>CTL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generality / Invariance</td>
<td>$G , f$</td>
<td>$AG , f$</td>
</tr>
<tr>
<td>Bounded response</td>
<td>$G(p \rightarrow X^n q)$</td>
<td>$AG(p \rightarrow (AX)^n q)$</td>
</tr>
<tr>
<td>Unbounded response</td>
<td>$G(p \rightarrow F q)$</td>
<td>$AG(p \rightarrow AF q)$</td>
</tr>
<tr>
<td>Infinitely often</td>
<td>$GF , p$</td>
<td>$AG , AF , p$</td>
</tr>
</tbody>
</table>
There are properties which cannot be expressed in both LTL and CTL

- $\neg G p / \neg AG p$, $\neg F p / \neg AF p$
- $G F p / AG AF p$ – both LTL and CTL
- $EF p$ – only CTL, but there is a workaround to check it in LTL!
- $F G p$ – only LTL
- $AG EF p$ – only CTL
- $CTL^*$ is a larger logic which allows combining quantified and unquantified temporal operators
Tool support

- **UPPAAL**: a subset of CTL, some LTL properties can be specified with automata
- **NuSMV**: both CTL and LTL
- **SPIN**: LTL