In supervisory control synthesis, there was a plant and requirements to be satisfied under the control of a blocking supervisor.

In LTL synthesis, there is no explicit plant, but only an LTL formula to be satisfied.
In supervisory control synthesis, there was a plant and requirements to be satisfied under the control of a blocking supervisor.

In LTL synthesis, there is no explicit plant, but only an LTL formula to be satisfied.

A controller interacts with an environment.
In supervisory control synthesis, there was a plant and requirements to be satisfied under the control of a blocking supervisor.

In LTL synthesis, there is no explicit plant, but only an LTL formula to be satisfied.

A controller interacts with an environment.

$x_1, ..., x_k$: input Boolean variables; the environment can assign any values to inputs on each step.
In supervisory control synthesis, there was a plant and requirements to be satisfied under the control of a blocking supervisor.

In LTL synthesis, there is no explicit plant, but only an LTL formula to be satisfied.

A controller interacts with an environment.

\[ x_1, \ldots, x_k: \text{input Boolean variables; the environment can assign any values to inputs on each step} \]

\[ y_1, \ldots, y_m: \text{output Boolean variables; the controller can assign any values to outputs on each step} \]
In supervisory control synthesis, there was a plant and requirements to be satisfied under the control of a blocking supervisor.

In LTL synthesis, there is no explicit plant, but only an LTL formula to be satisfied.

A controller interacts with an environment:

- $x_1, ..., x_k$: input Boolean variables; the environment can assign any values to inputs on each step.
- $y_1, ..., y_m$: output Boolean variables; the controller can assign any values to outputs on each step.

$f[x_1, ..., x_k, y_1, ..., y_m]$: LTL formula
LTL synthesis: problem formulation

- In supervisory control synthesis, there was a plant and requirements to be satisfied under the control of a blocking supervisor.
- In LTL synthesis, there is no explicit plant, but only an LTL formula to be satisfied.
- A controller interacts with an environment.
  - $x_1, ..., x_k$: input Boolean variables; the environment can assign any values to inputs on each step.
  - $y_1, ..., y_m$: output Boolean variables; the controller can assign any values to outputs on each step.
- $f[x_1, ..., x_k, y_1, ..., y_m]$: LTL formula.
- On each step, first the environment chooses the inputs, and then the controller chooses the outputs.
In supervisory control synthesis, there was a plant and requirements to be satisfied under the control of a blocking supervisor.

In LTL synthesis, there is no explicit plant, but only an LTL formula to be satisfied.

A controller interacts with an environment.

$x_1, ..., x_k$: input Boolean variables; the environment can assign any values to inputs on each step.

$y_1, ..., y_m$: output Boolean variables; the controller can assign any values to outputs on each step.

$f[x_1, ..., x_k, y_1, ..., y_m]$: LTL formula

On each step, first the environment chooses the inputs, and then the controller chooses the outputs.

LTL synthesis problem: synthesize a controller such that for all possible behaviors of the environment $f$ is satisfied.
Is the LTL synthesis problem solvable for the following formulae? If yes, how does the controller behave? If no, how should the environment behave to defeat any possible controller?

- $f = G(x \leftrightarrow y)$
Is the LTL synthesis problem solvable for the following formulae? If yes, how does the controller behave? If no, how should the environment behave to defeat any possible controller?

- \( f = G(x \leftrightarrow y) \) – yes; \( y := x \)
- \( f = G(x \rightarrow X y) \)
Is the LTL synthesis problem solvable for the following formulae? If yes, how does the controller behave? If no, how should the environment behave to defeat any possible controller?

- \( f = G(x \leftrightarrow y) \) – yes; \( y := x \)
- \( f = G(x \rightarrow X y) \) – yes; \( y := 1 \)
- \( f = G((X x) \rightarrow y) \) – no; the environment can choose the next different from the previous

- \( f = F(x \land y) \) – no; the environment can always set \( x := 0 \)
- \( f = F(x) \rightarrow F(x \land y) \) – yes; \( y := 1 \)
Is the LTL synthesis problem solvable for the following formulae? If yes, how does the controller behave? If no, how should the environment behave to defeat any possible controller?

- $f = G(x \leftrightarrow y)$ – yes; $y := x$
- $f = G(x \rightarrow X y)$ – yes; $y := 1$
- $f = G((X x) \rightarrow y)$ – yes; $y := 1$
- $f = G((X x) \leftrightarrow y)$
Is the LTL synthesis problem solvable for the following formulae? If yes, how does the controller behave? If no, how should the environment behave to defeat any possible controller?

- $f = G(x \leftrightarrow y)$ – yes; $y := x$
- $f = G(x \rightarrow X y)$ – yes; $y := 1$
- $f = G((X x) \rightarrow y)$ – yes; $y := 1$
- $f = G((X x) \leftrightarrow y)$ – no; the environment can choose the next $x$ different from the previous $y$
- $f = F(x \land y)$
Is the LTL synthesis problem solvable for the following formulae? If yes, how does the controller behave? If no, how should the environment behave to defeat any possible controller?

- $f = G(x \leftrightarrow y)$ – yes; $y := x$
- $f = G(x \rightarrow X y)$ – yes; $y := 1$
- $f = G((X x) \rightarrow y)$ – yes; $y := 1$
- $f = G((X x) \leftrightarrow y)$ – no; the environment can choose the next $x$ different from the previous $y$
- $f = F(x \land y)$ – no; the environment can always set $x := 0$
- $f = F x \rightarrow F(x \land y)$
Is the LTL synthesis problem solvable for the following formulae? If yes, how does the controller behave? If no, how should the environment behave to defeat any possible controller?

- $f = G(x \leftrightarrow y)$ – yes; $y := x$
- $f = G(x \rightarrow X y)$ – yes; $y := 1$
- $f = G((X x) \rightarrow y)$ – yes; $y := 1$
- $f = G((X x) \leftrightarrow y)$ – no; the environment can choose the next $x$ different from the previous $y$
- $f = F(x \land y)$ – no; the environment can always set $x := 0$
- $f = F x \rightarrow F(x \land y)$ – yes; $y := 1$
If the plant is finite-state, it is possible to encode it as an LTL formula

If $f_p$ describes the plant and $f_c$ are the requirements for the controller assuming that the plant submits to $f_p$, then it is sufficient to synthesize a controller for $f = f_p \rightarrow f_c$

The environment still can assign any possible values for inputs, but if it violates $f_p$, then the controller wins

Recall: in G4LTL-ST, $f_p$ was specified with ASSUME
To see how the LTL synthesis problem can be solved, we will look into the automata-theoretic approach to LTL model checking.
Runtime scenario: can we catch a specification violation while the system (or its model) is operating?

Assume that we have a Kripke structure of the system, then the monitor has access to atomic propositions on each step.

If we implement the monitor as a state machine, then it can have memory about previous assignments of atomic propositions.
Assume that we have a Kripke structure...

\[ f = G (\neg p) \]
Monitors: example of a safety automaton

Assume that we have a Kripke structure...

\[ f = G(\neg p) \]
Monitors: example of a safety automaton

Assume that we have a Kripke structure...

\[ f = \text{G}(\neg p) \]

State machine to check \( f \)? With guards on transitions and a rejecting state
Assume that we have a Kripke structure...

\[ f = G(\neg p) \]

State machine to check \( f \)? With guards on transitions and a rejecting state
An LTL formula $f$ is a **safety** formula, if all possible counterexamples to $f$ have a **finite prefix** such that every its infinite continuation is a counterexample.

Informally speaking, such properties state that “something bad” never happens.

Each safety property can be converted to a (possibly nondeterministic) safety automaton.

Safety automaton rejects an input sequence if it can visit a rejecting state while reading it.
Represent given properties as safety automata (1)

\[ f_1 = G(\neg p) \]

\[ f_2 = x \land X y \]

\[ f_3 = G(x \land X y) \]
Represent given properties as safety automata (1)

\[ f_1 = \mathbf{G}(\neg p) \]

\[ f_2 = x \land \mathbf{X}y \]

\[ f_3 = \mathbf{G}(x \land \mathbf{X}y) \]
Represent given properties as safety automata (1)

\[ f_1 = \mathbf{G}(\neg p) \]

\[ f_2 = x \land \mathbf{X} y \]

\[ f_3 = \mathbf{G}(x \land \mathbf{X} y) \]
Represent given properties as safety automata (1)

$$f_1 = G(\neg p)$$

$$f_2 = x \land X y$$

$$f_3 = G(x \land X y)$$
Represent given properties as safety automata (1)

\[ f_1 = \mathbf{G}(\neg p) \]

\[ f_2 = x \land \mathbf{X}y \]

\[ f_3 = \mathbf{G}(x \land \mathbf{X}y) \]
Represent given properties as safety automata (2)

\[ f_4 = G(x \rightarrow y \land X y) \]
Represent given properties as safety automata (2)

\[ f_4 = G(x \rightarrow y \land X y) \]

Diagram:
- State 0 with transitions:
  - From 0 to 1 on \( x \land y \)
  - From 0 to 2 on \( x \land y \)
  - From 0 to 0 on \( !x \land y \)
  - From 0 on loop on \( !x \)
- State 1 with transitions:
  - From 1 to 1 on \( x \land y \)
  - From 1 to 0 on \( x \land \neg y \)
  - From 1 on loop on \( !y \)
- State 2 with transition:
  - From 2 to 2 on \( \text{true} \)
Represent given properties as safety automata (2)

\[ f_4 = G(x \rightarrow y \land X y) \]

\[ f_5 = F y \]
Represent given properties as safety automata (2)

\[ f_4 = G(x \to y \land X y) \]

\[ f_5 = F y \]

Not a safety property!
$f_4 = \mathsf{G}(x \to y \land \mathsf{X} y)$
$f_4 = G(x \rightarrow y \land X y)$

Separate inputs from outputs
Transforming automata to safety games

\[ f_4 = G(x \rightarrow y \land X y) \]

Separate inputs from outputs
Transforming automata to safety games

\[ f_4 = \mathbb{G}(x \rightarrow y \land X y) \]

Separate inputs from outputs

\[
\begin{align*}
0 & \quad \text{!x} \\
1 & \quad x \& y \\
2 & \quad !y \\
\end{align*}
\]

\[
\begin{align*}
0 & \quad \text{true} \\
1 & \quad y \\
2 & \quad !y \\
\end{align*}
\]

\[
\begin{align*}
0 & \quad \text{true} \\
1 & \quad x \\
2 & \quad !y \\
\end{align*}
\]

\[
\begin{align*}
0 & \quad \text{true} \\
1 & \quad x \\
2 & \quad !y \\
\end{align*}
\]

\[
\begin{align*}
0 & \quad \text{true} \\
1 & \quad y \\
2 & \quad !y \\
\end{align*}
\]
Solving the safety game for the controller

Blue states are the ones where the controller makes choice. A winning strategy avoids red states.
Solving the safety game for the controller

Blue states are the ones where the controller makes choice.
A winning strategy avoids red states
Solving the safety game for the controller

Blue states are the ones where the controller makes choice.
A winning strategy avoids red states.
This solution applies **only** to safety LTL properties

Some simple reachability properties (like $f = F y$) can be handled by solving a reachability game instead (with the goal of the controller to reach a target state)
- This solution applies **only** to safety LTL properties.
- Some simple reachability properties (like $f = F y$) can be handled by solving a reachability game instead (with the goal of the controller to reach a target state).
- A safety game is the same as invariance game for non-rejecting states.
Some notes

- This solution applies **only** to safety LTL properties
- Some simple reachability properties (like $f = F y$) can be handled by solving a reachability game instead (with the goal of the controller to reach a target state)
- A safety game is the same as invariance game for non-rejecting states
- There is a simple graph algorithm to find the winning strategy (i.e. the required controller)
- This algorithm can also find that there is no winning strategy, i.e. the environment can always force the state machine to enter the rejecting state
Cylinder example (1)

- Binary position (home, ¬home) and binary control signal (fwd, ¬fwd)
- Specification for the plant (the position on the next turn is determined by the control signal): \( G(fwd \leftrightarrow X(\neg home)) \)
- We will require the controller to move the cylinder infinitely from one position to another: \( G(home \leftrightarrow X(\neg home)) \)
- Let’s put it together:

\[
  f = G(fwd \leftrightarrow X(\neg home)) \rightarrow G(home \leftrightarrow X(\neg home))
\]

- Is it a safety property?
Binary position (home, ¬home) and binary control signal (fwd, ¬fwd)

Specification for the plant (the position on the next turn is determined by the control signal): \( G(fwd \leftrightarrow X(\neg \text{home})) \)

We will require the controller to move the cylinder infinitely from one position to another: \( G(\text{home} \leftrightarrow X(\neg \text{home})) \)

Let’s put it together:

\[
f = G(fwd \leftrightarrow X(\neg \text{home})) \rightarrow G(\text{home} \leftrightarrow X(\neg \text{home}))\]

Is it a safety property? No! The environment can still violate plant assumptions even after the controller makes a mistake!
Cylinder example (2)

- How to solve the problem?

There is a more advanced method for non-safety formulae. If it is possible to convert the formula to a deterministic Büchi automaton, then the game-theoretical approach still applies with some modifications. Otherwise, every LTL property can be converted to a nondeterministic Büchi automaton, but then the solution is much more difficult.

In our particular case, we can modify the formula to make it safety!

\[ f' = (\text{home} \leftrightarrow X (\neg \text{home})) \]

\[ W \neg (\text{fwd} \leftrightarrow X (\neg \text{home})) \]

\[ W - \text{weak until:} \]

\[ x W y = (x U y) \lor (G x) \]
Cylinder example (2)

- How to solve the problem?
- Direct approach
  - There is a more advanced method for non-safety formulae
  - If it is possible to convert the formula to a deterministic Büchi automaton, then the game-theoretical approach still applies with some modifications
  - Otherwise, every LTL property can be converted to a nondeterministic Büchi automaton, but then the solution is much more difficult

\[ f' = (home \leftrightarrow X (\neg home)) \]
\[ W \neg (fwd \leftrightarrow X (\neg home)) \]
\[ W \text{– weak until:} \]
\[ x W y = (x U y) \lor (G x) \]
How to solve the problem?

Direct approach

- There is a more advanced method for non-safety formulae
- If it is possible to convert the formula to a deterministic Büchi automaton, then the game-theoretical approach still applies with some modifications
- Otherwise, every LTL property can be converted to a nondeterministic Büchi automaton, but then the solution is much more difficult
- In our particular case, we can modify the formula to make it safety!

The controller should satisfy the requirement until the environment violates plant assumptions:

\[ f' = (\text{home} \leftrightarrow X (\neg \text{home})) \]

\[ W \neg (f' \leftrightarrow X (\neg \text{home})) \]

\[ W - \text{weak until:} \]

\[ x W y = (x U y) \lor (G x) \]
Cylinder example (2)

- How to solve the problem?
- Direct approach
  - There is a more advanced method for non-safety formulae
  - If it is possible to convert the formula to a deterministic Büchi automaton, then the game-theoretical approach still applies with some modifications
  - Otherwise, every LTL property can be converted to a nondeterministic Büchi automaton, but then the solution is much more difficult
- In our particular case, we can modify the formula to make it safety!
  - The controller should satisfy the requirement until the environment violates plant assumptions

\[
 f' = (\text{home} \leftrightarrow X (\neg \text{home})) W \neg (\text{fwd} \leftrightarrow X (\neg \text{home})) W - \text{weak until:} \ x W y = (x U y) \lor (G x)
\]
Cylinder example (2)

- How to solve the problem?
- Direct approach
  - There is a more advanced method for non-safety formulae
  - If it is possible to convert the formula to a deterministic Büchi automaton, then the game-theoretical approach still applies with some modifications
  - Otherwise, every LTL property can be converted to a nondeterministic Büchi automaton, but then the solution is much more difficult
- In our particular case, we can modify the formula to make it safety!
  - The controller should satisfy the requirement until the environment violates plant assumptions
  - $f' = (\text{home} \leftrightarrow X(\neg \text{home})) \ W \neg (\text{fwd} \leftrightarrow X(\neg \text{home}))$
  - $W \dashv \text{weak until: } x \ W y = (x \ U y) \lor (G x)$
Safety automaton for the modified formula

Transform the automaton to a graph game and find the winning strategy for the controller.
