LTL synthesis

Igor Buzhinsky

igor.buzhinsky@gmail.com

August 20, 2020

LTL synthesis: problem formulation

- A controller interacts with an environment
- $x_1, \ldots, x_k$: input Boolean variables; the environment can assign any values to inputs on each step

LTL synthesis: problem formulation

- A controller interacts with an environment
- $x_1, \ldots, x_k$: input Boolean variables; the environment can assign any values to inputs on each step
- $y_1, \ldots, y_m$: output Boolean variables; the controller can assign any values to outputs on each step

LTL synthesis: problem formulation

- 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

- A controller interacts with an environment
- $x_1, \ldots, x_k$: input Boolean variables; the environment can assign any values to inputs on each step
- $y_1, \ldots, y_m$: output Boolean variables; the controller can assign any values to outputs on each step
- $f[x_1, \ldots, x_k, y_1, \ldots, y_m]$: LTL formula
- On each step, first the environment chooses the inputs, and then the controller chooses the outputs

LTL synthesis: problem formulation

- A controller interacts with an environment
- $x_1, \dots, x_k$: input Boolean variables; the environment can assign any values to inputs on each step
- $y_1, \dots, y_m$: output Boolean variables; the controller can assign any values to outputs on each step
- $f[x_1, \dots, x_k, y_1, \dots, y_m]$: 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

LTL synthesis: examples

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

LTL synthesis: examples

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

LTL synthesis: examples

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

LTL synthesis: examples

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

LTL synthesis: examples

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

LTL synthesis: examples

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

LTL synthesis: examples

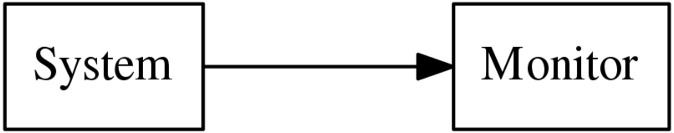
- Is the LTL synthesis problem solvable for the following formulas? If yes, how does the controller behave? If no, how should the environment behave to defeat any possible controller?
- $f = \mathbf{G}(x \leftrightarrow y)$ – yes; $y := x$
- $f = \mathbf{G}(x \rightarrow \mathbf{X}y)$ – yes; $y := 1$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

LTL synthesis: encoding the plant

- 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 $f = f_p \to f_c$
- The environment still can assign any possible values for inputs, but if it violates f_p, then the controller wins

To see how the LTL synthesis problem can be solved, we will look into the automata-theoretic approach to LTL model checking

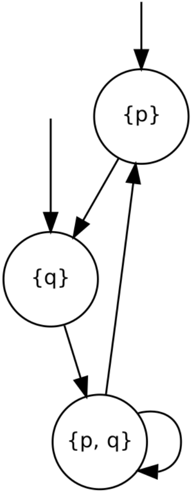
Monitors



- 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

Monitors: example of a safety automaton

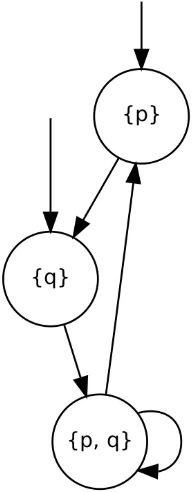
Assume that we have a Kripke structure...



Monitors: example of a safety automaton

Assume that we have a Kripke structure...

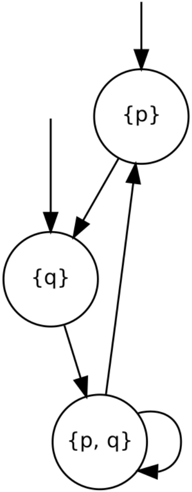
$f = \mathbf{G}(\neg p)$



Monitors: example of a safety automaton

Assume that we have a Kripke structure...

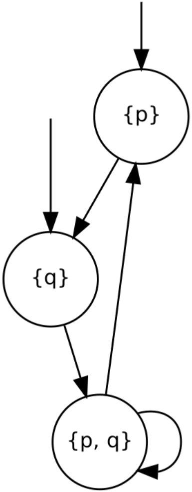
$$f = \mathbf{G}(\neg p)$$



State machine to check f? With guards on transitions and a rejecting state

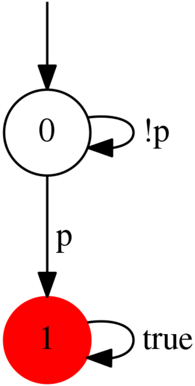
Monitors: example of a safety automaton

Assume that we have a Kripke structure...



$$f = \mathbf{G}(\neg p)$$

State machine to check f? With guards on transitions and a rejecting state

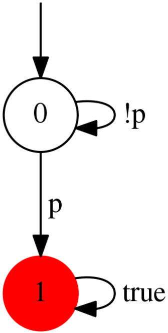


Safety LTL properties and safety automata

- 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)

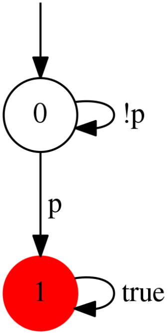
$f_1 = \mathbf{G}(\neg p)$



Represent given properties as safety automata (1)

$$f_2 = x \wedge \mathbf{X} y$$

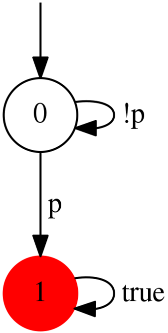
$$f_1 = \mathbf{G}(\neg p)$$

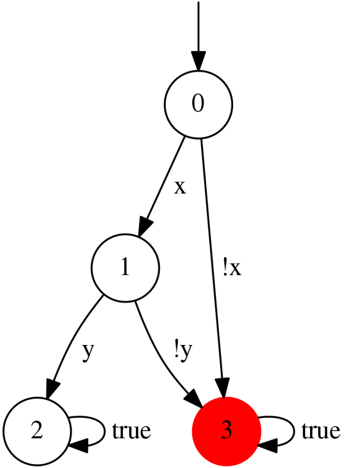


Represent given properties as safety automata (1)

$f_2 = x \wedge \mathbf{X} y$

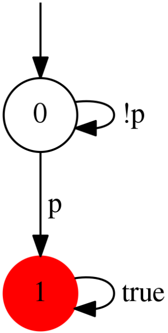
$f_1 = \mathbf{G}(\neg p)$



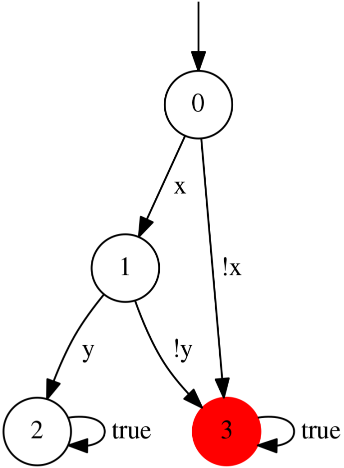


Represent given properties as safety automata (1)

$f_1 = \mathbf{G}(\neg p)$



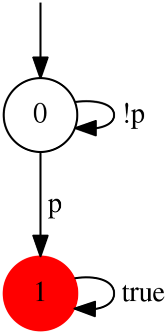
$f_2 = x \wedge \mathbf{X}y$



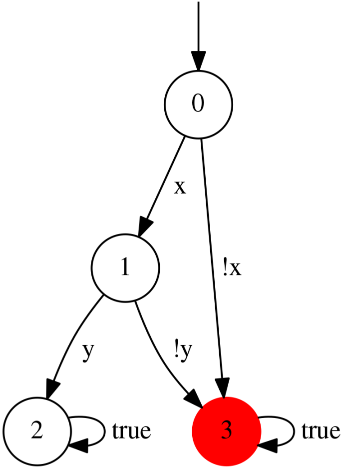
$f_3 = \mathbf{G}(x \wedge \mathbf{X}y)$

Represent given properties as safety automata (1)

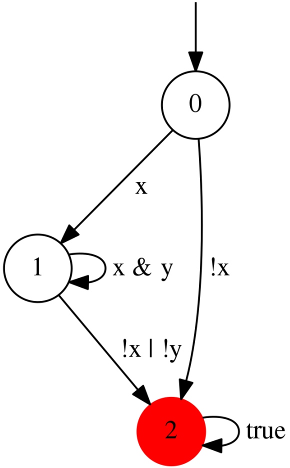
$f_1 = \mathbf{G}(\neg p)$



$f_2 = x \wedge \mathbf{X}y$



$f_3 = \mathbf{G}(x \wedge \mathbf{X}y)$

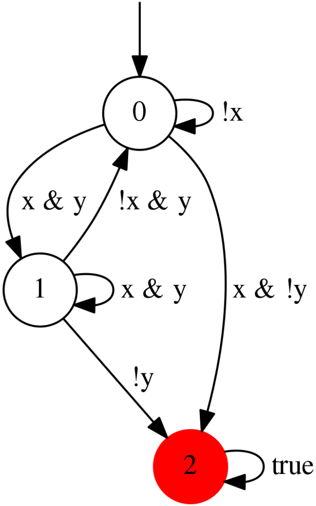


Represent given properties as safety automata (2)

$f_4 = \mathbf{G}(x \rightarrow y \wedge \mathbf{X} y)$

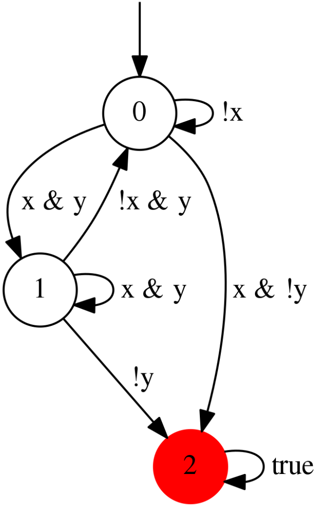
Represent given properties as safety automata (2)

$$f_4 = \mathbf{G}(x \to y \land \mathbf{X} y)$$



Represent given properties as safety automata (2)

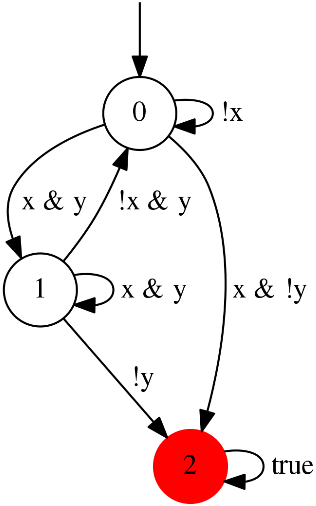
$f_4 = \mathbf{G}(x \to y \wedge \mathbf{X} y)$



$f_5 = \mathbf{F} y$

Represent given properties as safety automata (2)

$f_4 = \mathbf{G}(x \to y \wedge \mathbf{X} y)$



$f_5 = \mathbf{F} y$

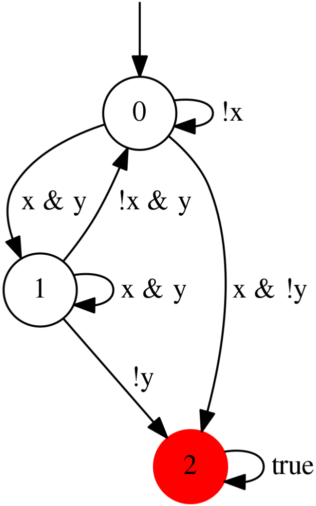
Not a safety property!

Transforming automata to safety games

$f_4 = \mathbf{G}(x \to y \wedge \mathbf{X} y)$

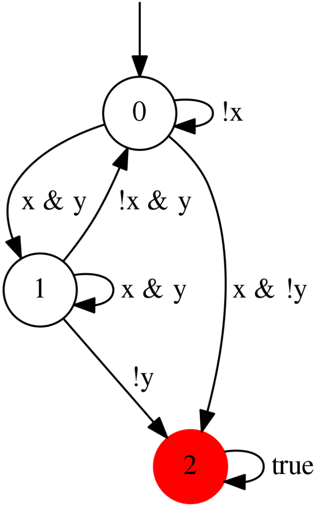
Transforming automata to safety games

$f_4 = \mathbf{G}(x \to y \wedge \mathbf{X}y)$



Transforming automata to safety games

$f_4 = \mathbf{G}(x \rightarrow y \wedge \mathbf{X}y)$

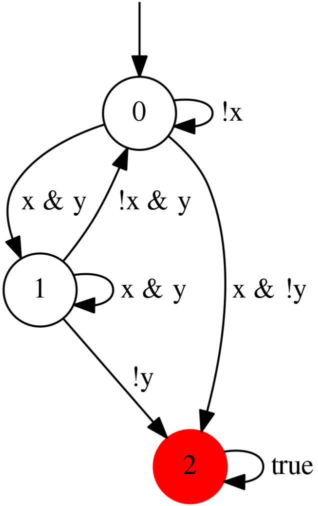


Separate inputs from outputs

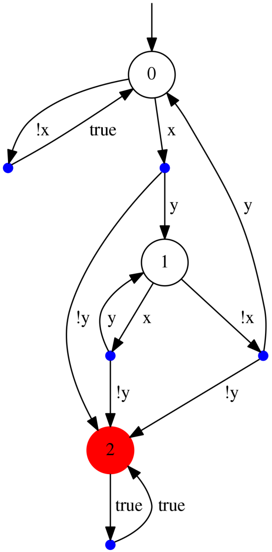
$\rightarrow$

Transforming automata to safety games

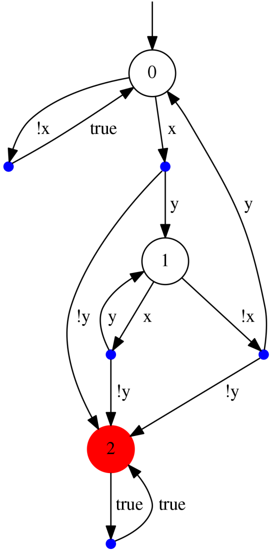
$f_4 = \mathbf{G}(x \to y \wedge \mathbf{X} y)$

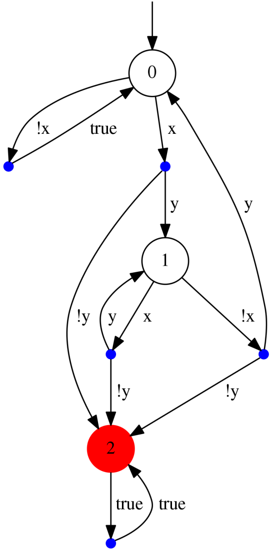


Separate inputs from outputs
$\rightarrow$



Solving the safety game for the controller

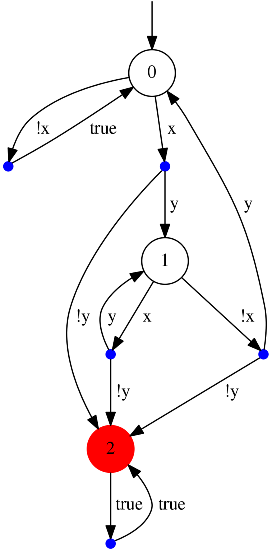




Blue states are the ones where the controller makes choice. A winning strategy avoids red states

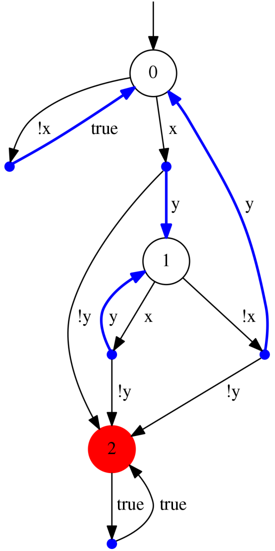
$\longrightarrow$

Solving the safety game for the controller



Blue states are the ones where the controller makes choice. A winning strategy avoids red states

$\rightarrow$



LTL synthesis algorithm for safety formulas

1. Transform the formula to a safety automaton
   - E.g., with the approach [Latvala T. Efficient model checking of safety properties. International SPIN Workshop on Model Checking of Software. Springer, Berlin, Heidelberg, 2003]

LTL synthesis algorithm for safety formulas

1. Transform the formula to a safety automaton
   - E.g., with the approach [Latvala T. Efficient model checking of safety properties. International SPIN Workshop on Model Checking of Software. Springer, Berlin, Heidelberg, 2003]
2. Separate the states where the controller and the environment make choices

LTL synthesis algorithm for safety formulas

1. Transform the formula to a safety automaton
   - E.g., with the approach [Latvala T. Efficient model checking of safety properties. International SPIN Workshop on Model Checking of Software. Springer, Berlin, Heidelberg, 2003]
2. Separate the states where the controller and the environment make choices
3. Compute the set of states where the controller loses
   1. Start from the violation state

LTL synthesis algorithm for safety formulas

1. Transform the formula to a safety automaton
   - E.g., with the approach [Latvala T. Efficient model checking of safety properties. International SPIN Workshop on Model Checking of Software. Springer, Berlin, Heidelberg, 2003]
2. Separate the states where the controller and the environment make choices
3. Compute the set of states where the controller loses
   1. Start from the violation state
   2. If the environment can make a move to a controller-losing state, this state is also controller-losing

LTL synthesis algorithm for safety formulas

1. Transform the formula to a safety automaton
   - E.g., with the approach [Latvala T. Efficient model checking of safety properties. International SPIN Workshop on Model Checking of Software. Springer, Berlin, Heidelberg, 2003]
2. Separate the states where the controller and the environment make choices
3. Compute the set of states where the controller loses
   1. Start from the violation state
   2. If the environment can make a move to a controller-losing state, this state is also controller-losing
   3. If all possible local moves of the controller lead to controller-losing states, this state is also controller-losing

LTL synthesis algorithm for safety formulas

1. Transform the formula to a safety automaton
   - E.g., with the approach [Latvala T. Efficient model checking of safety properties. International SPIN Workshop on Model Checking of Software. Springer, Berlin, Heidelberg, 2003]
2. Separate the states where the controller and the environment make choices
3. Compute the set of states where the controller loses
   1. Start from the violation state
   2. If the environment can make a move to a controller-losing state, this state is also controller-losing
   3. If all possible local moves of the controller lead to controller-losing states, this state is also controller-losing
   4. Apply these expansion rules until no further states can be added

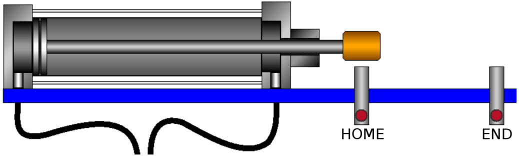
LTL synthesis algorithm for safety formulas

1. Transform the formula to a safety automaton
   - E.g., with the approach [Latvala T. Efficient model checking of safety properties. International SPIN Workshop on Model Checking of Software. Springer, Berlin, Heidelberg, 2003]
2. Separate the states where the controller and the environment make choices
3. Compute the set of states where the controller loses
   1. Start from the violation state
   2. If the environment can make a move to a controller-losing state, this state is also controller-losing
   3. If all possible local moves of the controller lead to controller-losing states, this state is also controller-losing
   4. Apply these expansion rules until no further states can be added
4. Select any controller strategy that does not make transitions to controller-losing states (if this is impossible, then no controller exists that solves this LTL synthesis probem)

LTL synthesis for reachability properties

- This previous solution applies **only** to safety LTL properties
- Some simple reachability properties (like $f = \mathbf{F}x$) can be handled by solving a reachability game instead (with the goal of the controller to reach a target state)
  1. Compute the set of controller-winning states, starting from the target state
  2. Memorize controller's transitions that lead to controller-winning states (they will form the solution)
  3. Expand controller-winning states while possible
  4. If the initial state is controller-winning, then we have the solution, otherwise the problem is unsolveable

Cylinder example (1)

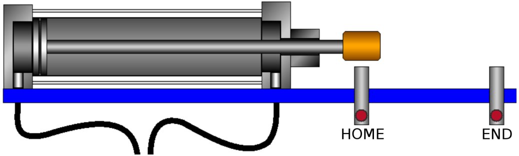


- Binary position (home, $\neg$home) and binary control signal (fwd, $\neg$fwd)
- Specification for the plant (the position on the next turn is determined by the control signal): $\mathbf{G}(\text{fwd} \leftrightarrow \mathbf{X}(\neg\text{home}))$
- We will require the controller to move the cylinder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

$$f = \mathbf{G}(\text{fwd} \leftrightarrow \mathbf{X}(\neg\text{home})) \rightarrow \mathbf{G}(\text{home} \leftrightarrow \mathbf{X}(\neg\text{home}))$$

- Is it a safety property?

Cylinder example (1)



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

$$f = \mathbf{G}(\text{fwd} \leftrightarrow \mathbf{X}(\neg\text{home})) \rightarrow \mathbf{G}(\text{home} \leftrightarrow \mathbf{X}(\neg\text{home}))$$

- 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?

Cylinder example (2)

- How to solve the problem?
- Direct approach
  - There is a more advanced method for non-safety formulas
  - 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

Cylinder example (2)

- How to solve the problem?
- Direct approach
  - There is a more advanced method for non-safety formulas
  - 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!

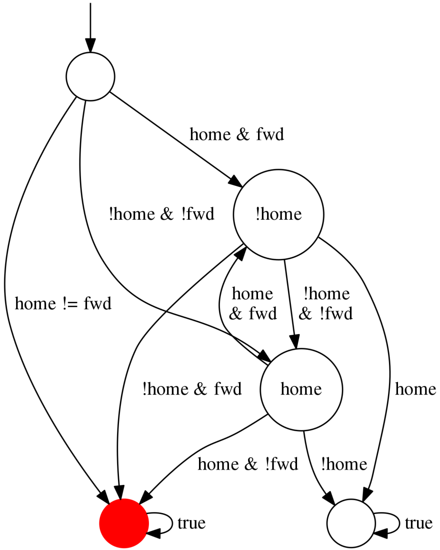
Cylinder example (2)

- How to solve the problem?
- Direct approach
  - There is a more advanced method for non-safety formulas
  - 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

Cylinder example (2)

- How to solve the problem?
- Direct approach
  - There is a more advanced method for non-safety formulas
  - 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
  - $f' = (\mathrm{home} \leftrightarrow \mathbf{X}(\neg\mathrm{home}))\mathbf{W}\neg(\mathrm{fwd} \leftrightarrow \mathbf{X}(\neg\mathrm{home}))$
  - $\mathbf{W}$ is weak until: $x\mathbf{W}y = (x\mathbf{U}y) \vee (\mathbf{G}x)$

Safety automaton for the modified formula



Exercise: transform the automaton to a graph game and find the winning strategy for the controller

Literature

- Pnueli A., Rosner R. On the synthesis of a reactive module. Proc. Symposium on Principles of Programming Languages (POPL '89), 1989, pp. 179–190
- Rosner R. Modular synthesis of reactive systems. Ph.D. dissertation, Weizmann Institute of Science, 1992
- Latvala T. Efficient model checking of safety properties. International SPIN Workshop on Model Checking of Software. Springer, Berlin, Heidelberg, 2003
- Schewe S., Finkbeiner B. Bounded synthesis. In K.S. Namjoshi, T. Yoneda, T. Higashino, Y. Okamura (eds.) ATVA, LNCS, vol. 4762, pp. 474–488. Springer, 2007