Formal verification. Model checking. Formal specifications: LTL, CTL. NuSMV and nuXmv model checkers

Igor Buzhinsky

igor.buzhinsky@gmail.com

August 19, 2020

Introduction

Formal verification vs. testing

Specifications

- Functional: input-output relationship (what the system must do)
- Non-functional: qualities of the system, such as safety, security, performance, response time, usability, testability, maintainability, extensibility, scalability...
- Informal and formal (e.g., formulas, standardized diagrams)

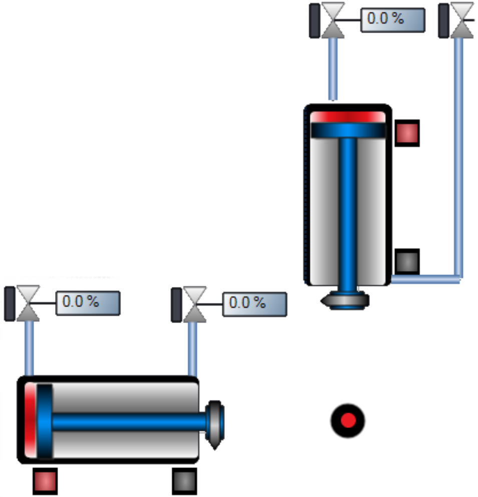
Testing

- Finite, often non-exhaustive number of scenarios to consider
- Test cases can be written manually or generated automatically

Formal verification

- Formal model: unambiguous, often condensed and simplified view on certain aspects of the system to be verified
- Formal specifications are needed
- Often exhaustive checking (under the assumptions of the formal models)

Example of a system: TwoCylinders



- Two cylinders and a tray
- If a workpiece is placed on a tray, it can be pushed away by a cylinder
- The cylinders will collide if they are both extended almost entirely

Formal verification methods

- Static analysis
  - Analyze the system without executing it
  - For example, can detect that the last assignment is unreachable:

```
retracted = position == 0;
extended = position == 10;
if (retracted && extended) { error = True; }
```

- Model checking
  - Mostly for checking functional specifications
  - Prove or disprove that the cylinders cannot collide by examining all possible system behaviors
- Theorem proving
  - Prove a wider range of properties
  - For a particular cylinder collision algorithm, prove that if it works for $i > 1$ cylinders, then it will also work for $i + 1$ cylinders

Model checking

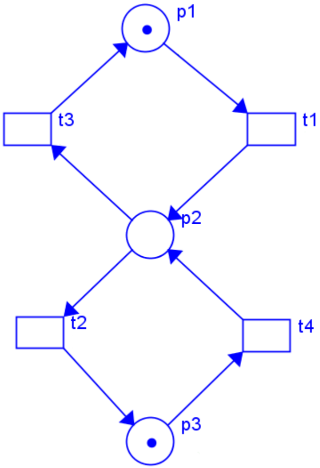
Model checking: overview

- Requires a **formal model** of the behavior of the system
- Mathematically, this model is often represented as some form of a state machine
  - Visual tools to do this, e.g., UPPAAL
- There are formal languages that allow specifying formal models textually
  - E.g., NuSMV, nuXmv, SPIN

- Similarly, **temporal logics** are formal languages that specify properties to be verified
- These properties often examine **infinite model behaviors**

- Due to checking behaviors, the specifications that are checked are often functional
- With carefully created models and temporal formulas, it is also possible to verify response time, safety and security

Model example: Petri nets

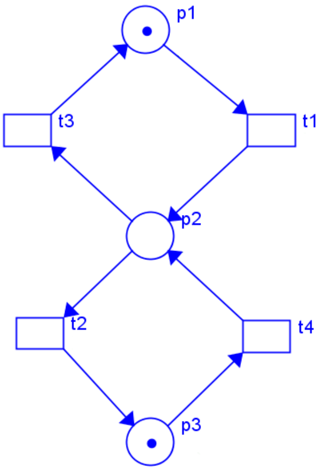


- Locations (circles) can contain tokens (dots)
- Each location can contain 0 or more tokens
- The initial configuration is shown on the slide
- Each transition (box) can move a token from its input location to its output location
- If multiple transitions can execute, then the one to be executed is selected **nondeterministically**

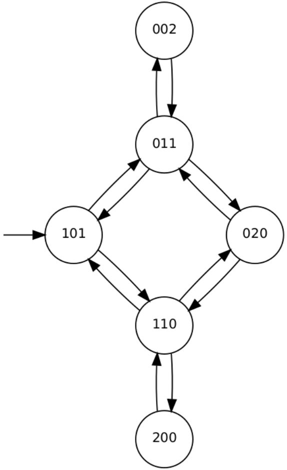
State (reachability) graph of a system

- 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



A Petri net

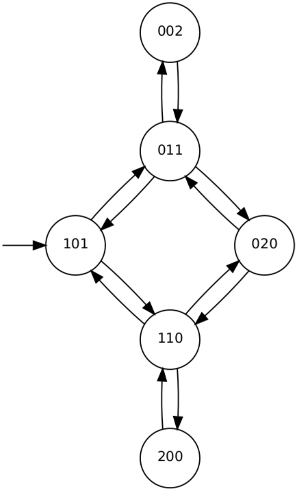


State graph, state $= p_1 p_2 p_3$

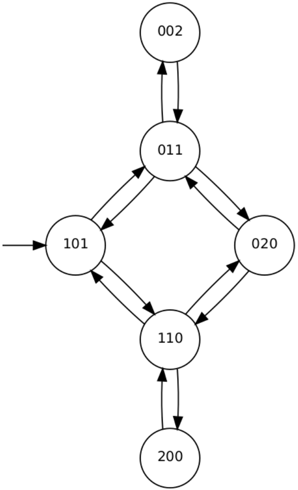
Kripke structures

- Formalization of a state graph
- Let AP be 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**
- $I \subseteq S$ is a set of **initial states**
- $T \subseteq S \times S$ is a **transition relation**
- $L : S \to 2^{\mathrm{AP}}$ is a **labeling function**
- **No deadlock assumption:** $\forall s \in S \; \exists s' \in S : (s, s') \in T$, i.e., it is possible to proceed to somewhere from any state

State graph interpreted as a Kripke structure

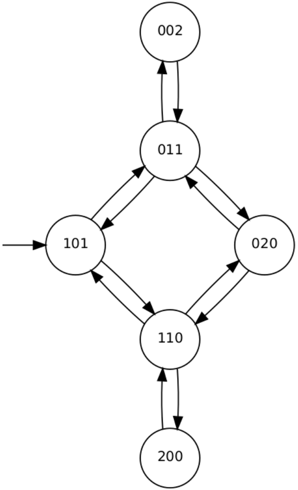


State graph interpreted as a Kripke structure



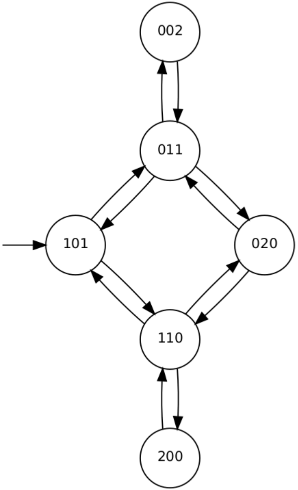
- AP =

State graph interpreted as a Kripke structure



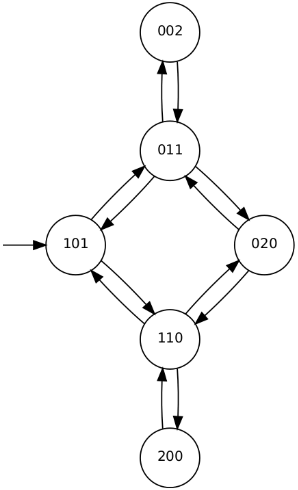
- AP $= \{$"$p_i = j$" $\mid 1 \leq i \leq 3, 0 \leq j \leq 2\}$
- S:

State graph interpreted as a Kripke structure



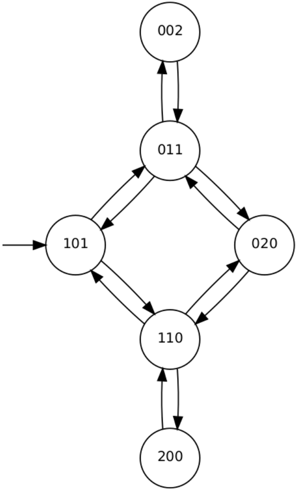
- $\mathrm{AP} = \{\text{“}p_i = j\text{”} \mid 1 \leq i \leq 3, 0 \leq j \leq 2\}$
- S: nodes of this graph
- $I \subseteq S =$

State graph interpreted as a Kripke structure



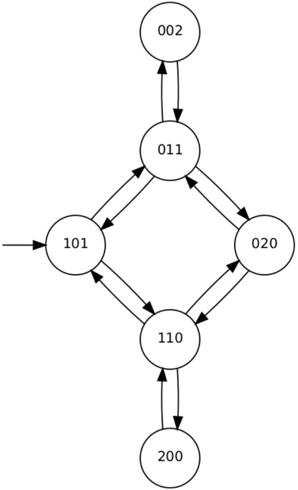
- $\mathrm{AP} = \{\text{“}p_i = j\text{”} \mid 1 \leq i \leq 3, 0 \leq j \leq 2\}$
- S: nodes of this graph
- $I \subseteq S = \{101\}$

State graph interpreted as a Kripke structure



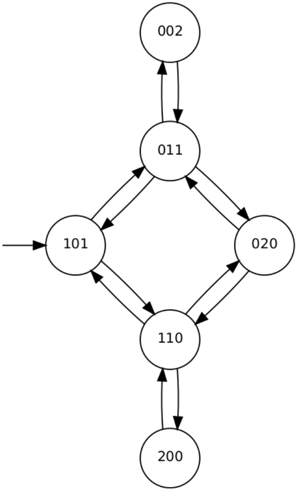
- AP $= \{\text{“}p_i = j\text{”} \mid 1 \leq i \leq 3, 0 \leq j \leq 2\}$
- S: nodes of this graph
- $I \subseteq S = \{101\}$
- $T \subseteq S \times S$:

State graph interpreted as a Kripke structure



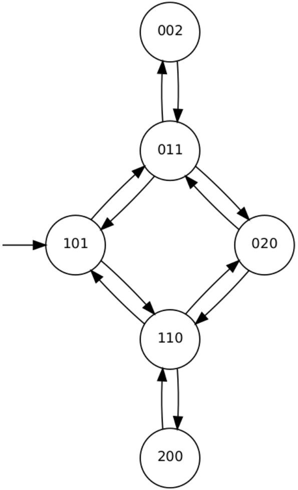
- $\mathrm{AP} = \{\text{“}p_i = j\text{”} \mid 1 \leq i \leq 3, 0 \leq j \leq 2\}$
- S: nodes of this graph
- $I \subseteq S = \{101\}$
- $T \subseteq S \times S$: edges of the graph, e.g., $(002, 011)$, $(011, 002)$, ...
- $L : S \to 2^{\mathrm{AP}}$:

State graph interpreted as a Kripke structure



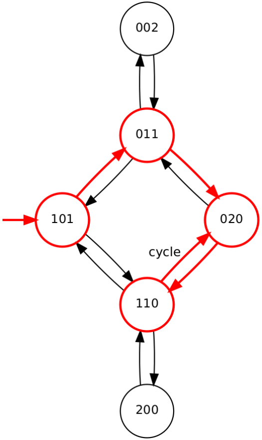
- $\text{AP} = \{\text{"}p_i = j\text{"} \mid 1 \leq i \leq 3, 0 \leq j \leq 2\}$
- S: nodes of this graph
- $I \subseteq S = \{101\}$
- $T \subseteq S \times S$: edges of the graph, e.g., $(002, 011)$, $(011, 002)$, ...
- $L : S \rightarrow 2^{\text{AP}}$: token assignments (markings) in each state, e.g., $L(020) = \{\text{"}p_1 = 0\text{"}, \text{"}p_2 = 2\text{"}, \text{"}p_3 = 0\text{"}\}$

State graph interpreted as a Kripke structure



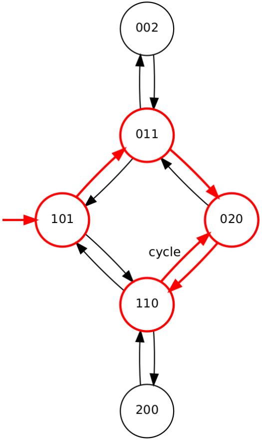
- AP $= \{\text{“}p_i = j\text{”} \mid 1 \leq i \leq 3, 0 \leq j \leq 2\}$
- S: nodes of this graph
- $I \subseteq S = \{101\}$
- $T \subseteq S \times S$: edges of the graph, e.g., $(002, 011)$, $(011, 002)$, ...
- $L : S \to 2^{\text{AP}}$: token assignments (markings) in each state, e.g., $L(020) = \{\text{“}p_1 = 0\text{”}, \text{“}p_2 = 2\text{”}, \text{“}p_3 = 0\text{”}\}$
- **Specifications** can be interpreted as predicates over Kripke structures

System behaviors are paths in Kripke structures

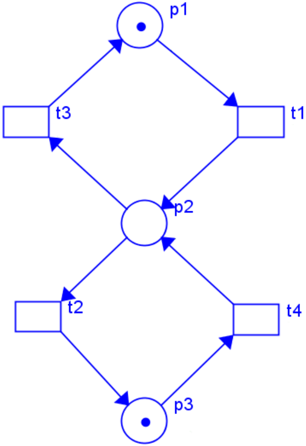


Infinite paths are common in formal verification. This is the reason why deadlocks are undesirable

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 model?

Linear temporal logic (LTL)

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: $\{p, q\}, \{p\}, \{\}, \text{cycle}(\{q\}, \{p, q\})$

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 this example)

Linear temporal logic (LTL)

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

Linear temporal logic (LTL)

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

- **Temporal operators**

Linear temporal logic (LTL)

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

- **Temporal operators**
  - **G**: **g**lobally (always), e.g., $\mathbf{G}(p \wedge \neg q)$ means "in each element of the sequence, $p \wedge \neg q$ holds"

Linear temporal logic (LTL)

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

- **Temporal operators**
  - **G**: **g**lobally (always), e.g., $\mathbf{G}(p \wedge \neg q)$ means "in each element of the sequence, $p \wedge \neg q$ holds"
  - **F**: in the **f**uture, e.g., $\mathbf{F}(p \wedge \neg q)$ means "for some element of the sequence, $p \wedge \neg q$ holds"

Linear temporal logic (LTL)

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

- **Temporal operators**
  - **G**: **g**lobally (always), e.g., $\mathbf{G}(p \wedge \neg q)$ means "in each element of the sequence, $p \wedge \neg q$ holds"
  - **F**: in the **f**uture, e.g., $\mathbf{F}(p \wedge \neg q)$ means "for some element of the sequence, $p \wedge \neg q$ holds"
  - **X**: on the ne**x**t step, e.g., $\mathbf{X}(p \wedge \neg q)$ means "$p \wedge \neg q$ holds for the second element of the sequence"

Linear temporal logic (LTL)

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

- **Temporal operators**
  - **G**: **g**lobally (always), e.g., $\mathbf{G}(p \wedge \neg q)$ means "in each element of the sequence, $p \wedge \neg q$ holds"
  - **F**: in the **f**uture, e.g., $\mathbf{F}(p \wedge \neg q)$ means "for some element of the sequence, $p \wedge \neg q$ holds"
  - **X**: on the ne**x**t step, e.g., $\mathbf{X}(p \wedge \neg q)$ means "$p \wedge \neg q$ holds for the second element of the sequence"
  - **U**: **u**ntil (binary operator), e.g., $p\mathbf{U}q$ means "q must happen at some step, and the sequence must satisfy p until (non-inclusive) q happens"

Examples of LTL formulas

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

- $f_1 = \mathbf{G}p$

Examples of LTL formulas

- 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
- $f_2 = \mathbf{F}(\neg p \wedge \neg q)$

Examples of LTL formulas

- 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
- $f_2 = \mathbf{F}(\neg p \wedge \neg q)$ – paths 1, 3
- $f_3 = p\mathbf{U}(\neg p \wedge \neg q)$

Examples of LTL formulas

- 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
- $f_2 = \mathbf{F}(\neg p \wedge \neg q)$ – paths 1, 3
- $f_3 = p\mathbf{U}(\neg p \wedge \neg q)$ – paths 1, 3

- Temporal operators can be applied to arbitrary LTL formulas!
- $f_4 = \mathbf{XXXX}p$

Examples of LTL formulas

- 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
- $f_2 = \mathbf{F}(\neg p \wedge \neg q)$ – paths 1, 3
- $f_3 = p\mathbf{U}(\neg p \wedge \neg q)$ – paths 1, 3

- Temporal operators can be applied to arbitrary LTL formulas!
- $f_4 = \mathbf{XXXX}p$ ("on the fifth step") – paths 1, 2, 3
- $f_5 = \mathbf{FG}(p \wedge q)$

Examples of LTL formulas

- 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
- $f_2 = \mathbf{F}(\neg p \wedge \neg q)$ – paths 1, 3
- $f_3 = p\mathbf{U}(\neg p \wedge \neg q)$ – paths 1, 3

- Temporal operators can be applied to arbitrary LTL formulas!
- $f_4 = \mathbf{XXXX}p$ ("on the fifth step") – paths 1, 2, 3
- $f_5 = \mathbf{FG}(p \wedge q)$ ("globally from some point") – path 2
- $f_6 = \mathbf{GF}(p \wedge q)$

Examples of LTL formulas

- 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
- $f_2 = \mathbf{F}(\neg p \wedge \neg q)$ – paths 1, 3
- $f_3 = p\mathbf{U}(\neg p \wedge \neg q)$ – paths 1, 3

- Temporal operators can be applied to arbitrary LTL formulas!
- $f_4 = \mathbf{XXXX}p$ ("on the fifth step") – paths 1, 2, 3
- $f_5 = \mathbf{FG}(p \wedge q)$ ("globally from some point") – path 2
- $f_6 = \mathbf{GF}(p \wedge q)$ ("infinitely often") – paths 1, 2, 3
- $f_7 = \mathbf{G}(p \rightarrow \mathbf{X}q)$

Examples of LTL formulas

- 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
- $f_2 = \mathbf{F}(\neg p \wedge \neg q)$ – paths 1, 3
- $f_3 = p\mathbf{U}(\neg p \wedge \neg q)$ – paths 1, 3

- Temporal operators can be applied to arbitrary LTL formulas!
- $f_4 = \mathbf{XXXX}p$ ("on the fifth step") – paths 1, 2, 3
- $f_5 = \mathbf{FG}(p \wedge q)$ ("globally from some point") – path 2
- $f_6 = \mathbf{GF}(p \wedge q)$ ("infinitely often") – paths 1, 2, 3
- $f_7 = \mathbf{G}(p \to \mathbf{X}q)$ ("p is always followed by q") – paths 2, 3

LTL: some simplification and equivalence rules

- $\mathbf{GG}f = \mathbf{G}f$
- $\mathbf{FF}f = \mathbf{F}f$

LTL: some simplification and equivalence rules

- $\mathbf{GG}f = \mathbf{G}f$
- $\mathbf{FF}f = \mathbf{F}f$

- $\mathbf{GX}f = \mathbf{XG}f$
- $\mathbf{FX}f = \mathbf{XF}f$

LTL: some simplification and equivalence rules

- $\mathbf{GG}f = \mathbf{G}f$
- $\mathbf{FF}f = \mathbf{F}f$

- $\mathbf{GX}f = \mathbf{XG}f$
- $\mathbf{FX}f = \mathbf{XF}f$

- $\neg\mathbf{G}(f) = \mathbf{F}(\neg f)$
- $\neg\mathbf{F}(f) = \mathbf{G}(\neg f)$

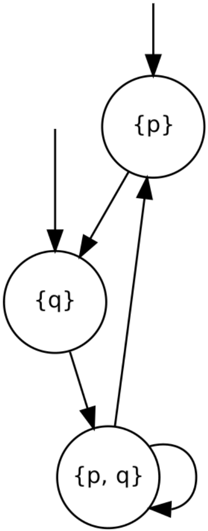
- Also remember that Boolean rules can be applied to Boolean subformulas, e.g., $\mathbf{G}(f \to g)$ is equivalent to $\mathbf{G}(\neg f \lor g)$

LTL model checking: 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

LTL model checking: 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 formulas are satisfied by the KS on the right? Why?

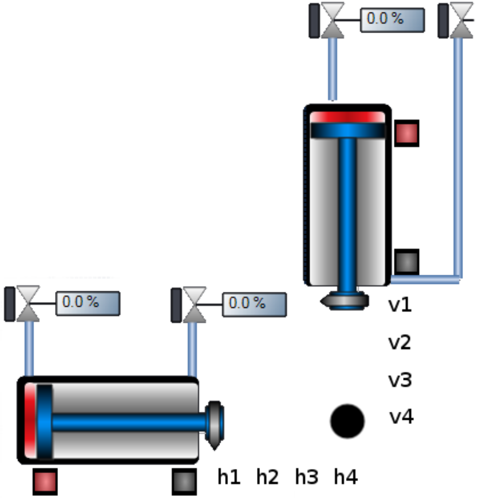
- $f_1 = \mathbf{G}p$
- $f_2 = \mathbf{F}(\neg p \wedge \neg q)$
- $f_3 = p\mathbf{U}(\neg p \wedge \neg q)$
- $f_4 = \mathbf{XXXX}p$
- $f_5 = \mathbf{FG}(p \wedge q)$
- $f_6 = \mathbf{GF}(p \wedge q)$
- $f_7 = \mathbf{G}(p \rightarrow \mathbf{X}q)$

Quiz answers

Quiz answers

- Only $f_6 = \mathbf{GF}(p \wedge q)$

Formal model of the TwoCylinders 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

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:

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

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: $\mathbf{G}(h_1 \vee h_2 \vee h_3 \vee h_4)$, $\mathbf{G}(v_1 \vee v_2 \vee v_3 \vee v_4)$
- Cylinder can't have more than one position:
  $\mathbf{G}(\neg(h_1 \wedge h_2) \wedge \neg(h_1 \wedge h_3) \wedge \neg(h_1 \wedge h_4) \wedge \neg(h_2 \wedge h_3) \wedge \neg(h_2 \wedge h_4) \wedge \neg(h_3 \wedge h_4))$,
  $\mathbf{G}(\neg(v_1 \wedge v_2) \wedge \neg(v_1 \wedge v_3) \wedge \neg(v_1 \wedge v_4) \wedge \neg(v_2 \wedge v_3) \wedge \neg(v_2 \wedge v_4) \wedge \neg(v_3 \wedge v_4))$
- 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: $\mathbf{G}(h_1 \vee h_2 \vee h_3 \vee h_4)$, $\mathbf{G}(v_1 \vee v_2 \vee v_3 \vee v_4)$
- Cylinder can't have more than one position:
  $\mathbf{G}(\neg(h_1 \wedge h_2) \wedge \neg(h_1 \wedge h_3) \wedge \neg(h_1 \wedge h_4) \wedge \neg(h_2 \wedge h_3) \wedge \neg(h_2 \wedge h_4) \wedge \neg(h_3 \wedge h_4))$,
  $\mathbf{G}(\neg(v_1 \wedge v_2) \wedge \neg(v_1 \wedge v_3) \wedge \neg(v_1 \wedge v_4) \wedge \neg(v_2 \wedge v_3) \wedge \neg(v_2 \wedge v_4) \wedge \neg(v_3 \wedge v_4))$
- If a cylinder is fully extended, then there is no workpiece: $\mathbf{G}(h_4 \vee v_4 \to \neg w)$
- ...

- 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: $\mathbf{G}\neg(h_4 \land v_4)$

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: $\mathbf{G}\neg(h_4 \wedge v_4)$
- 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: $\mathbf{G}\neg(h_4 \wedge v_4)$
- 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: $\mathbf{G}\neg(h_4 \land v_4)$
- 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):

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: $\mathbf{G}\neg(h_4 \wedge v_4)$
- When a workpiece appears, it must be eventually pushed away: $\mathbf{G}(w \to \mathbf{F}\neg w)$
- Cylinders iterate (each new workpiece is pushed by a different cylinder): $\mathbf{G}((h_4 \wedge (\mathbf{X}\neg h_4) \wedge \mathbf{F}w) \to \mathbf{X}((\neg w \wedge \neg v_4 \wedge \neg h_4)\mathbf{U}(w \wedge (w\mathbf{U}(v_4 \wedge \neg h_4)))))$, and the same for the other cylinder

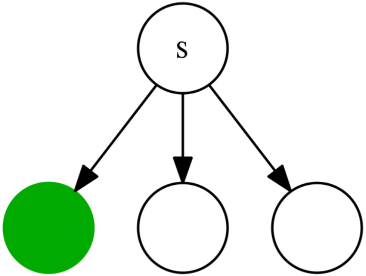
Computation tree logic (CTL)

Computation tree logic (CTL)

- In LTL, there is always an implicit quantification over all paths starting in initial states
- In **CTL**, all temporal operators are annotated with quantifiers

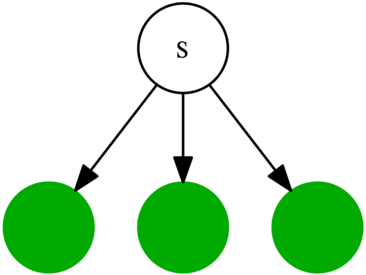
- CTL formulas characterize not infinite sequences, but rather states of the Kripke structure
- A Kripke structure satisfies a CTL formula, if **all its initial states** satisfy this formula
- Let s be a state of the KS, then $s \models f$ means s satisfies f

CTL: temporal operator **EX**



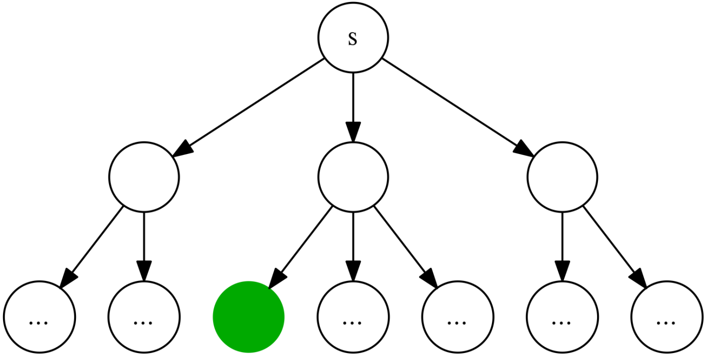
- $s \models \mathbf{EX}(f)$ ("exists next"): in some successor of s, f holds

CTL: temporal operator **AX**



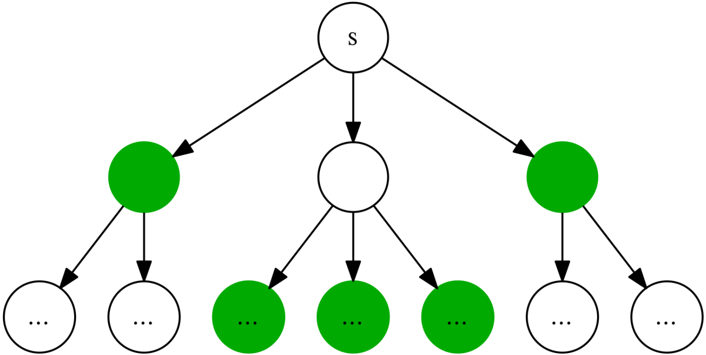
- $s \models \mathbf{AX}(f)$ ("for all next"): in all successors of s, f holds

CTL: temporal operator **EF**

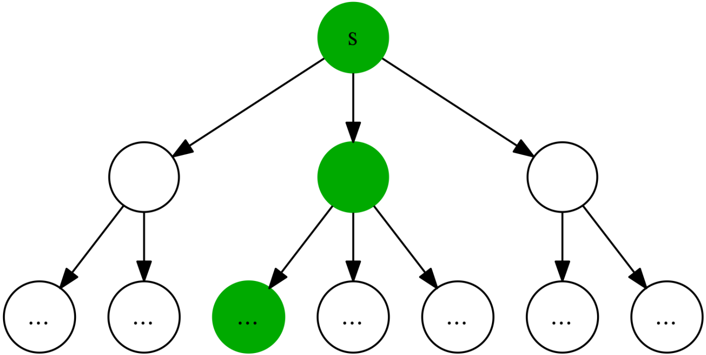


- $s \models \mathbf{EF}(f)$ ("exists in the future"): there exists a path starting in s such that f becomes valid at some point of this path

CTL: temporal operator **AF**

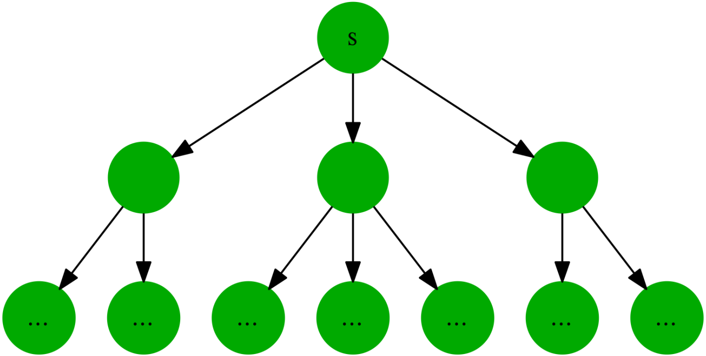


- $s \models \mathbf{AF}(f)$ ("for all in the future"): for all possible paths starting in s, f becomes true at some point



- $s \models \mathbf{EG}(f)$ ("exists globally"): there exists a path starting in s such that f holds at every state along this path

CTL: temporal operator **AG**



- $s \models \mathbf{AG}(f)$ ("for all globally"): for all possible paths starting in s, f is always true

CTL: temporal operators **EU** and **AU**

- $s \models f\,\mathbf{EU}\,g$ ("exists until"): there exists a path starting in s such that f holds until (non-inclusive) g, and g eventually happens
- $s \models f\,\mathbf{AU}\,g$ ("for all until"): for all possible paths starting in s, f holds until (non-inclusive) g, and g eventually happens

Are these formulas syntactically correct in LTL or CTL?

- $p \wedge \neg q$

Are these formulas syntactically correct in LTL or CTL?

- $p \land \neg q$ – both LTL and CTL

Are these formulas syntactically correct in LTL or CTL?

- $p \wedge \neg q$ – both LTL and CTL
- $\mathbf{AX}(p \to \mathbf{F}q)$

Are these formulas syntactically correct in LTL or CTL?

- $p \land \neg q$ – both LTL and CTL
- $\mathbf{AX}(p \to \mathbf{F}q)$ – incorrect

Are these formulas syntactically correct in LTL or CTL?

- $p \wedge \neg q$ – both LTL and CTL
- $\mathbf{AX}(p \rightarrow \mathbf{F}q)$ – incorrect
- $\mathbf{FXAG}q$

Are these formulas syntactically correct in LTL or CTL?

- $p \wedge \neg q$ – both LTL and CTL
- $\mathbf{AX}(p \to \mathbf{F}q)$ – incorrect
- $\mathbf{FXAG}q$ – incorrect

Are these formulas syntactically correct in LTL or CTL?

- $p \land \neg q$ – both LTL and CTL
- $\mathbf{AX}(p \rightarrow \mathbf{F}q)$ – incorrect
- $\mathbf{FXAG}q$ – incorrect
- $\mathbf{EXAG}q$

Are these formulas syntactically correct in LTL or CTL?

- $p \wedge \neg q$ – both LTL and CTL
- $\mathbf{AX}(p \rightarrow \mathbf{F}q)$ – incorrect
- $\mathbf{FXAG}q$ – incorrect
- $\mathbf{EXAG}q$ – CTL

Are these formulas syntactically correct in LTL or CTL?

- $p \wedge \neg q$ – both LTL and CTL
- $\mathbf{AX}(p \to \mathbf{F}q)$ – incorrect
- $\mathbf{FXAG}q$ – incorrect
- $\mathbf{EXAG}q$ – CTL
- $\mathbf{EX}\neg\mathbf{AG}q$

Are these formulas syntactically correct in LTL or CTL?

- $p \wedge \neg q$ – both LTL and CTL
- $\mathbf{AX}(p \to \mathbf{F}q)$ – incorrect
- $\mathbf{FXAG}q$ – incorrect
- $\mathbf{EXAG}q$ – CTL
- $\mathbf{EX}\neg\mathbf{AG}q$ – CTL

Are these formulas syntactically correct in LTL or CTL?

- $p \wedge \neg q$ – both LTL and CTL
- $\mathbf{AX}(p \rightarrow \mathbf{F}q)$ – incorrect
- $\mathbf{FXAG}q$ – incorrect
- $\mathbf{EXAG}q$ – CTL
- $\mathbf{EX}\neg\mathbf{AG}q$ – CTL
- $\mathbf{G}(p \rightarrow \mathbf{XXF}q)$

Are these formulas syntactically correct in LTL or CTL?

- $p \land \neg q$ – both LTL and CTL
- $\mathbf{AX}(p \to \mathbf{F}q)$ – incorrect
- $\mathbf{FXAG}q$ – incorrect
- $\mathbf{EXAG}q$ – CTL
- $\mathbf{EX}\neg\mathbf{AG}q$ – CTL
- $\mathbf{G}(p \to \mathbf{XXF}q)$ – LTL

Are these formulas syntactically correct in LTL or CTL?

- $p \wedge \neg q$ – both LTL and CTL
- $\mathbf{AX}(p \rightarrow \mathbf{F}q)$ – incorrect
- $\mathbf{FXAG}q$ – incorrect
- $\mathbf{EXAG}q$ – CTL
- $\mathbf{EX}\neg\mathbf{AG}q$ – CTL
- $\mathbf{G}(p \rightarrow \mathbf{XXF}q)$ – LTL
- $(\mathbf{AXAXAXAX}p)\mathbf{U}(\mathbf{EF}\neg p)$

Are these formulas syntactically correct in LTL or CTL?

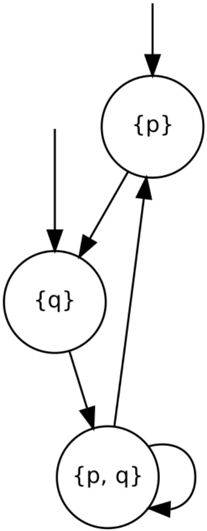
- $p \wedge \neg q$ – both LTL and CTL
- $\mathbf{AX}(p \rightarrow \mathbf{F}q)$ – incorrect
- $\mathbf{FXAG}q$ – incorrect
- $\mathbf{EXAG}q$ – CTL
- $\mathbf{EX}\neg\mathbf{AG}q$ – CTL
- $\mathbf{G}(p \rightarrow \mathbf{XXF}q)$ – LTL
- $(\mathbf{AXAXAXAX}p)\mathbf{U}(\mathbf{EF}\neg p)$ – incorrect

CTL model checking: example

- KS satisfies the CTL formula iff all its initial states satisfy it

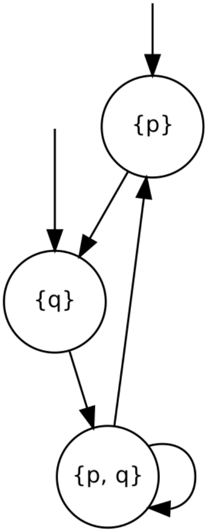
CTL model checking: example

- KS satisfies the CTL formula iff all its initial states satisfy it
- Which of these CTL formulas are satisfied by the KS on the right? Why?
- $f_1 = \mathbf{AG}p$
- $f_2 = \mathbf{AG}(p \vee q)$
- $f_3 = \mathbf{AF}(p \wedge q)$
- $f_4 = \mathbf{EF}(\neg p \wedge \neg q)$
- $f_5 = \mathbf{AXAXAXAX}p$
- $f_6 = \mathbf{EFEG}(p \wedge q)$
- $f_7 = \mathbf{EGEF}(p \wedge q)$
- $f_8 = \mathbf{AG}(p \to \mathbf{AX}q)$



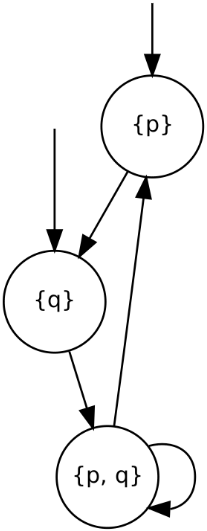
CTL model checking: example

- KS satisfies the CTL formula iff all its initial states satisfy it
- Which of these CTL formulas are satisfied by the KS on the right? Why?
- $f_1 = \mathbf{AG}p$
- $f_2 = \mathbf{AG}(p \vee q)$
- $f_3 = \mathbf{AF}(p \wedge q)$
- $f_4 = \mathbf{EF}(\neg p \wedge \neg q)$
- $f_5 = \mathbf{AXAXAXAX}p$
- $f_6 = \mathbf{EFEG}(p \wedge q)$
- $f_7 = \mathbf{EGEF}(p \wedge q)$
- $f_8 = \mathbf{AG}(p \rightarrow \mathbf{AX}q)$
- Answer:

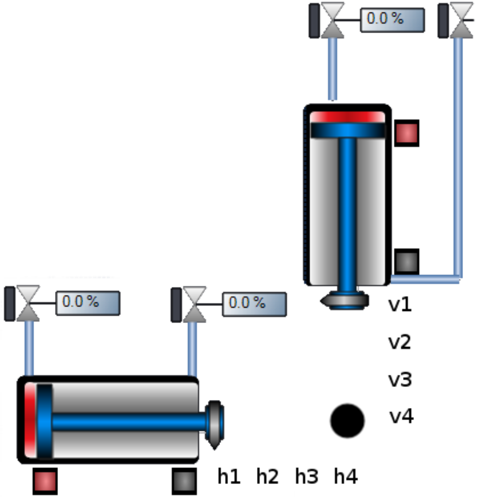


CTL model checking: example

- KS satisfies the CTL formula iff all its initial states satisfy it
- Which of these CTL formulas are satisfied by the KS on the right? Why?
- $f_1 = \mathbf{AG}p$
- $f_2 = \mathbf{AG}(p \vee q)$
- $f_3 = \mathbf{AF}(p \wedge q)$
- $f_4 = \mathbf{EF}(\neg p \wedge \neg q)$
- $f_5 = \mathbf{AXAXAXAX}p$
- $f_6 = \mathbf{EFEG}(p \wedge q)$
- $f_7 = \mathbf{EGEF}(p \wedge q)$
- $f_8 = \mathbf{AG}(p \to \mathbf{AX}q)$
- Answer: f_2, f_3, f_6, f_7



CTL model checking: two cylinders



- Atomic propositions: $h_1..h_4, v_1..v_4, w$
- **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

Quiz answers

- If a cylinder is fully extended, then there is no workpiece: $\mathbf{AG}(h_4 \lor v_4 \to \neg w)$
- Cylinders do not collide: $\mathbf{AG}\neg(h_4 \land v_4)$
- When a workpiece appears, it must be eventually pushed away: $\mathbf{AG}(w \to \mathbf{AF}\neg w)$
- Cylinders iterate: ¯_(ツ)_/¯

Common specification types ("patterns")

Name	LTL	CTL
Generality / Invariance	$\mathbf{G}f$	$\mathbf{AG}f$
Bounded response	$\mathbf{G}(p \rightarrow \mathbf{X}^n q)$	$\mathbf{AG}(p \rightarrow (\mathbf{AX})^n q)$
Unbounded response	$\mathbf{G}(p \rightarrow \mathbf{F}q)$	$\mathbf{AG}(p \rightarrow \mathbf{AF}q)$
Infinitely often	$\mathbf{GF}p$	$\mathbf{AGAF}p$

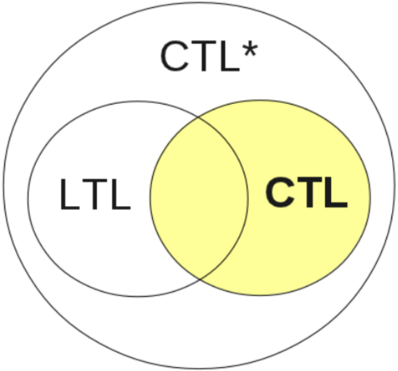
Common specification types ("patterns")

Name	LTL	CTL
Generality / Invariance	$\mathbf{G}f$	$\mathbf{AG}f$
Bounded response	$\mathbf{G}(p \to \mathbf{X}^n q)$	$\mathbf{AG}(p \to (\mathbf{AX})^n q)$
Unbounded response	$\mathbf{G}(p \to \mathbf{F}q)$	$\mathbf{AG}(p \to \mathbf{AF}q)$
Infinitely often	$\mathbf{GF}p$	$\mathbf{AGAF}p$

- These are lucky cases, but unfortunately, transforming one logic to the other cannot be always done by appending/removing **A** or **E**
- Sometimes such a transformation is impossible – see the next slide

There are properties which cannot be expressed in both LTL and CTL

- $\mathbf{G}p/\mathbf{AG}p$, $\mathbf{F}p/\mathbf{AF}p$, $\mathbf{GF}p/\mathbf{AGAF}p$ – both LTL and CTL
- $\mathbf{EF}p$ – only CTL, but there is a workaround to check it in LTL!
- $\mathbf{FG}p$ – only LTL
- $\mathbf{AGEF}p$ – only CTL
- CTL* is a larger logic which allows combining quantified and unquantified temporal operators



NuSMV and nuXmv model checkers

NuSMV

- Open-source symbolic model checker
- Supports LTL and CTL
- Can be downloaded here: `http://nusmv.fbk.eu/`
- Command-line tool, models are specified in text files
- If an LTL specification is false, the corresponding counterexample can be visualized with the tool
  `https://github.com/igor-buzhinsky/nusmv_counterexample_visualizer`

NuSMV: example

```
MODULE main
VAR
    p: boolean;
    c: 0..10;
    e: {value_a, value_b, value_c};
DEFINE
    c_plus_1 := c + 1;
ASSIGN
    init(c) := 0;
    next(c) := c_plus_1 mod 10;
    init(p) := FALSE;
    next(p) := next(c = 5) | p;

CTLSPEC AG(c != 10)
LTLSPEC G(p -> X(p))
```

- `VAR`: variable declarations (Boolean, integer, enumeration)

NuSMV: example

```
MODULE main
VAR
    p: boolean;
    c: 0..10;
    e: {value_a, value_b, value_c};
DEFINE
    c_plus_1 := c + 1;
ASSIGN
    init(c) := 0;
    next(c) := c_plus_1 mod 10;
    init(p) := FALSE;
    next(p) := next(c = 5) | p;

CTLSPEC AG(c != 10)
LTLSPEC G(p -> X(p))
```

- `VAR`: variable declarations (Boolean, integer, enumeration)
- `DEFINE`: create an alias for a sub-expression

NuSMV: example

```
MODULE main
VAR
    p: boolean;
    c: 0..10;
    e: {value_a, value_b, value_c};
DEFINE
    c_plus_1 := c + 1;
ASSIGN
    init(c) := 0;
    next(c) := c_plus_1 mod 10;
    init(p) := FALSE;
    next(p) := next(c = 5) | p;

CTLSPEC AG(c != 10)
LTLSPEC G(p -> X(p))
```

- `VAR`: variable declarations (Boolean, integer, enumeration)
- `DEFINE`: create an alias for a sub-expression
- `ASSIGN`/`init`: specify an initial value (or a set of values) of a variable

NuSMV: example

```
MODULE main
VAR
    p: boolean;
    c: 0..10;
    e: {value_a, value_b, value_c};
DEFINE
    c_plus_1 := c + 1;
ASSIGN
    init(c) := 0;
    next(c) := c_plus_1 mod 10;
    init(p) := FALSE;
    next(p) := next(c = 5) | p;

CTLSPEC AG(c != 10)
LTLSPEC G(p -> X(p))
```

- `VAR`: variable declarations (Boolean, integer, enumeration)
- `DEFINE`: create an alias for a sub-expression
- `ASSIGN`/`init`: specify an initial value (or a set of values) of a variable
- `ASSIGN`/`next`: specify the next value (or a set of values) of a variable
- `next` assignments can refer to next values of other variables and current values of all variables

NuSMV: cylinder

```
MODULE CYLINDER(fwd, back)
VAR
    pos: 0..5;
ASSIGN
    init(pos) := 0;
    next(pos) := fwd ? next_pos : back ? prev_pos : pos;
DEFINE
    next_pos := pos < 5 ? (pos + 1) : pos;
    prev_pos := pos > 0 ? (pos - 1) : pos;
    home := pos = 0;
    end := pos = 5;
```

- Modules can have inputs (in the declaration), and their variables and definitions can be interpreted as outputs
- C-style choice operator ?:

NuSMV: controller

```
MODULE CONTROLLER(home, end)
VAR
    state: {moving_fwd, moving_back};
ASSIGN
    init(state) := moving_fwd;
    next(state) := case
        home: moving_fwd;
        end: moving_back;
        TRUE: state;
    esac;
DEFINE
    fwd := state = moving_fwd;
    back := state = moving_back;
```

- Example of explicit state machine modeling

NuSMV: closed-loop composition

```
MODULE main
VAR
    -- this is the way to write comments, by the way
    cyl: CYLINDER(ctr.fwd, ctr.back);
    ctr: CONTROLLER(cyl.home, cyl.end);

LTLSPEC G F cyl.end -- TRUE
LTLSPEC G F cyl.home -- TRUE
```

- **Synchronous**: all the modules make a step together!

NuSMV: closed-loop composition

```
MODULE main
VAR
    -- this is the way to write comments, by the way
    cyl: CYLINDER(ctr.fwd, ctr.back);
    ctr: CONTROLLER(cyl.home, cyl.end);

LTLSPEC G F cyl.end -- TRUE
LTLSPEC G F cyl.home -- TRUE
```

- **Synchronous**: all the modules make a step together!
- How to model asynchronous interaction?

NuSMV: closed-loop composition

```
MODULE main
VAR
    -- this is the way to write comments, by the way
    cyl: CYLINDER(ctr.fwd, ctr.back);
    ctr: CONTROLLER(cyl.home, cyl.end);

LTLSPEC G F cyl.end -- TRUE
LTLSPEC G F cyl.home -- TRUE
```

- **Synchronous**: all the modules make a step together!
- How to model asynchronous interaction? – introduce execution permissions; no permission = keep the state of the module unchanged

nuXmv model checker

- Successor of NuSMV
- Mostly backward compatible with NuSMV (i.e., you can run it on the same models)
- Can be downloaded here: `https://es-static.fbk.eu/tools/nuxmv/`
- New features
  - Infinite-state integer and real variables
  - Verification of real-time models

Literature

- Baier, C., & Katoen, J. P. (2008). Principles of model checking. MIT press
- Clarke, E. M., Grumberg, O., & Peled, D. (1999). Model checking. MIT press
- Schneider, K. (2013). Verification of reactive systems: formal methods and algorithms. Springer Science & Business Media
- Cimatti, A., et al. NuSMV 2: An opensource tool for symbolic model checking. International Conference on Computer Aided Verification. 2002
- Cimatti, A., et al. Extending nuXmv with timed transition systems and timed temporal properties. International Conference on Computer Aided Verification. 2019
- NuSMV tutorial: `http://nusmv.fbk.eu/NuSMV/tutorial/`