Theory of concurrency

Lecture 2

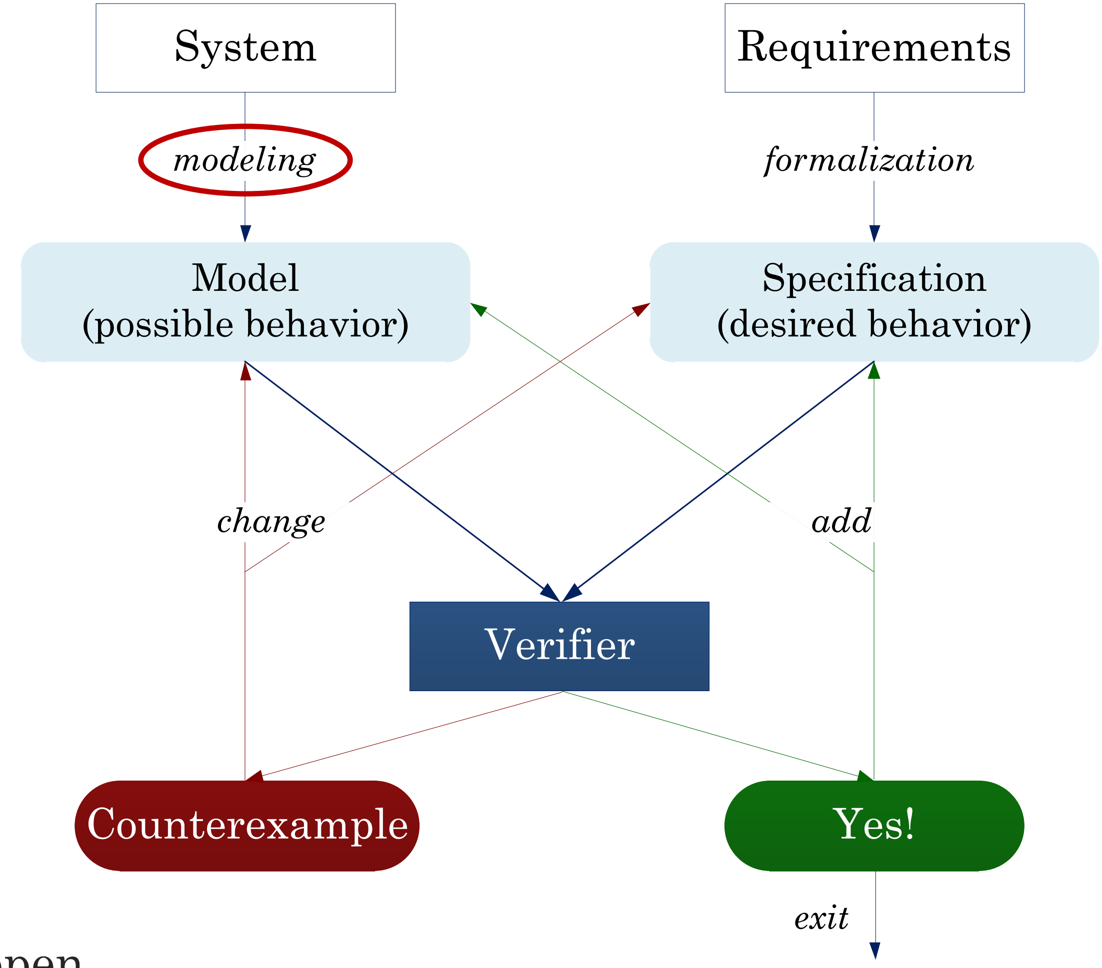
Model checking

Details

System modeling

Model checking process

- *Modeling*
  - Translating a description of
    a software system to the formal description
    suitable for automatic verification.
    - Compilation, translation.
    - Abstraction.
- Specification
  - Formulating software properties systems
    for verification
    - Logic: temporal, dynamic,
      epistemic, deontic and many others.
    - Completeness of the specification.
      - Safety: nothing bad will ever happen.
      - Liveliness: something good is definitely happen.
- Verification



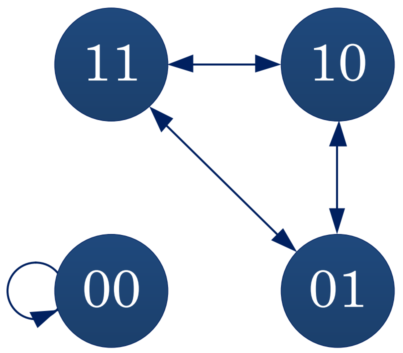
System modeling

- *Formal model* of a system for verification
  - Everything you need
  - Nothing extra
    - *Collision avoidance protocols*:
      - times and places
      - not names of machines
    - *Telecommunications*:
      - message exchange
      - not message content
    - *Control systems*:
      - control signals
      - not physical values

System modeling

- Reactive systems
  - Interaction with the environment
  - Infinitely long work

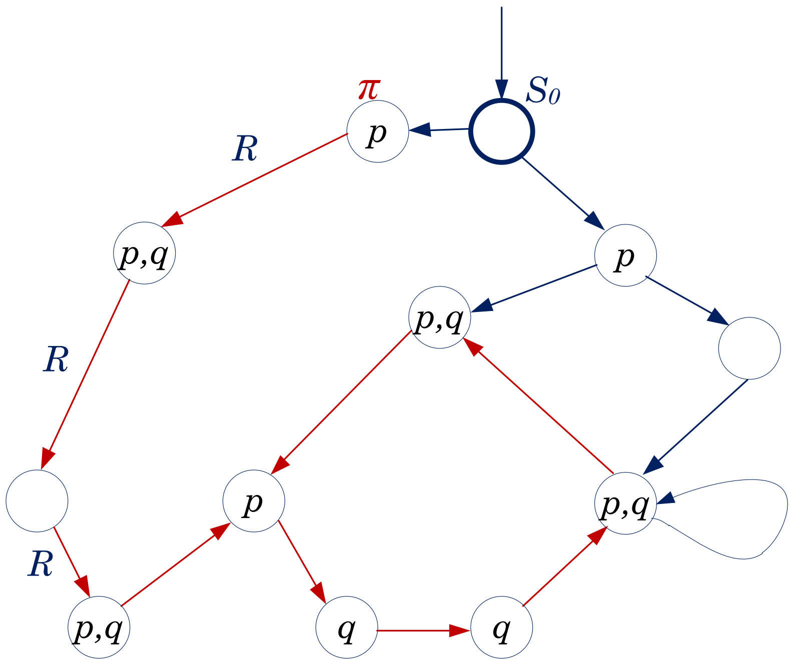
  - *A state*
    - instant description of a system, the value of system variables at a given time.
  - *A state transition*
    - state change
    - a state before and a state after
  - *A computation*
    - is an infinite sequence of states where each state is obtained from the previous state by some transition.



System modeling

- *Kripke structure*
  - Special transition graph
  - The paths in the graph correspond to system computations
  - Simple and universal model
- *Concurrent systems*
  - Program text or chart
  - Types of systems
    - synchronous, asynchronous, with shared variables, exchanging messages, etc.
  - Described by first–order predicate logic formulas
    - Formulas are correctly translated into the Kripke structure

System modeling



- *A set of atomic propositions* AP
  - $x = 13$.
  - Elevator doors open.
  - The sun is shining.
- *Kripke structure* $M = (S, S_0, R, L)$
  - S — finite set of states;
  - $S_0 \subseteq S$ — set of initial states;
  - $R \subseteq S \times S$ — total transition relation: for each $s \in S$ there exists $s' \in S$ such that $R(s, s')$;
  - $L: S \to 2^{AP}$ — a function that labels each state with the set of atomic propositions true in that state.
- *Path* in model M from state s
  - (in-)finite sequence of model states $\pi = s_0, s_1, s_2, \ldots$, such that
    - $s_0 = s$ and for all $i \geq 0$ it is true that $R(s_i, s_{i+1})$.

FOL concurrent system representation

- *First-Order Logic*
  - Predicate and functional symbols with fixed interpretation
  - Logical connectives $\neg$, $\wedge$, $\vee$, $\rightarrow$
  - Quantifiers $\forall$, $\exists$
- *Description of parallel systems*
  - $V = \{v_0, v_1, v_2, \dots v_n\}$ – *system variables*
  - D – *domain of interpretation*
  - A *valuation* for V is a function that associates a value in D with each variable v in V.
- *State* s is a valuation: $s: V \rightarrow D$.
  - A formula represents *the set of all states* in which it is true.
  - A formula that is true *exactly* on a given valuation
    - $V = \{v_0, v_1, v_2\}$, $D = \{0, 1, 2\}$
    - $s = \{v_0 \leftarrow 0, v_1 \leftarrow 1, v_2 \leftarrow 2\}$:
      - $\varphi: (v_0 = 0) \wedge (v_1 = 1) \wedge (v_2 = 2)$

0,1,2

$\psi: (v_0 = 0) \wedge (v_1 = 1) \vee (v_2 = 2)$

0,0,2 1,0,2 2,0,2 0,1,0

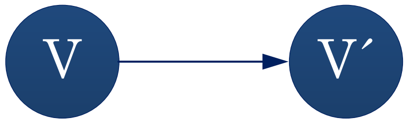
0,1,2 1,1,2 2,1,2 0,1,1

0,2,2 1,2,2 2,2,2

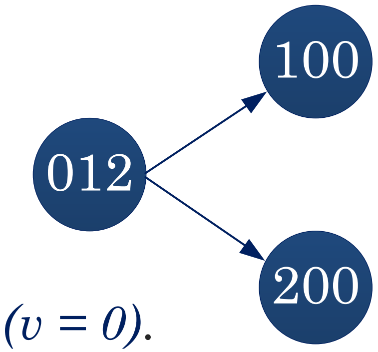
- *Atomic propositions AP*
  - $v = d$, $v \in V$, $d \in D$: $v = d$ is true in state s, if $s(v)=d$.

FOL concurrent system representation

- *Transitions* between states
  - Formulas to represent sets of ordered pairs of states
  - Let V be system variables, V' — copies of system variables
    - V — current state variables
    - V' — next state variables



  - A valuation of V and V' encodes a transition
    - The value is encoded by a first-order formula
    - $s = \{v_0 \leftarrow 0,\ v_1 \leftarrow 1,\ v_2 \leftarrow 2\}$,
      $s' = \{v'_0 \leftarrow 1,\ v'_1 \leftarrow 0,\ v'_2 \leftarrow 0\}$,
      $s'' = \{v'_0 \leftarrow 2,\ v'_1 \leftarrow 0,\ v'_2 \leftarrow 0\}$:
      - $(v_0 = 0) \wedge (v_1 = 1) \wedge (v_2 = 2) \wedge ((v'_0 = 1) \vee (v'_0 = 2)) \wedge \forall v \in \{v'_1, v'_2\} : (v = 0)$.



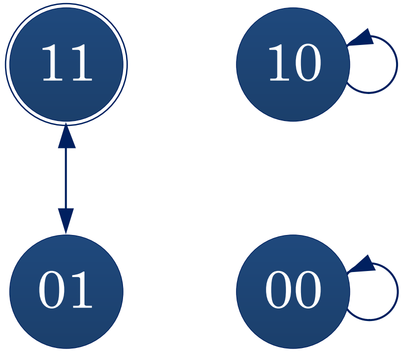
  - ***R(V,V')*** — first order formula representing the transition relation R.

Concurrent system representation

- The Kripke structure $M = (S, S_0, R, L)$ from first-order formulas $\boldsymbol{S_0}$ and $\boldsymbol{R(V, V')}$.
  - <u>States</u> S — the set of all valuations of the variables V;
  - <u>Initial states</u> S_0 — the set of all valuations s_0 of the variables V that satisfy the formula $\boldsymbol{S_0}$.
  - <u>Transitions</u> R — $R(s, s')$ are true for all s, s' such that the formula $\boldsymbol{R}$ is true for the valuations $s(v)$ and $s(v')$ for all $v \in V$ and $v' \in V'$.
    - Some state s may have no successor.
      - Modify the relation R so that $R(s, s)$ is true.
  - <u>Labeling function</u> $L: S \to 2^{AP}$ — for all states s the set $L(s)$ are all atomic propositions true in s.
    - If the variable v is Boolean, then we write $v \in L(s)$ or $v \notin L(s)$.

Concurrent system representation

- Example
  - Let $V = \{x, y\}$, $D = \{0, 1\}$
  - Valuations: $(d_1, d_2) \in D{\times}D$
  - Transition: $x := (x+y)(mod2)$,
    Initial state: $x = 1$ и $y = 1$.
- First order representation:
  - $\boldsymbol{S_0(x, y)} \equiv (x = 1) \wedge (y = 1)$
  - $\boldsymbol{R(x, y, x', y')} \equiv x' = (x+y)(mod2) \wedge (y' = y)$
- Kripke structure $M = (S, S_0, R, L)$
  - $S = D \times D$
  - $S_0 = \{(1,1)\}$
  - $R = \{((1,1), (0,1)), ((0,1), (1,1)), ((1,0), (1,0)), ((0,0), (0,0))\}$
  - $L((0,0)) = \{x = 0, y = 0\}$, $L((0,1)) = \{x = 0, y = 1\}$, $L((1,0)) = \{x = 1, y = 0\}$, $L((1,1)) = \{x = 1, y = 1\}$
- The *only* path from initial state and the only computation of the system:
  - $(1,1), (0,1), (1,1), (0,1)\ldots$

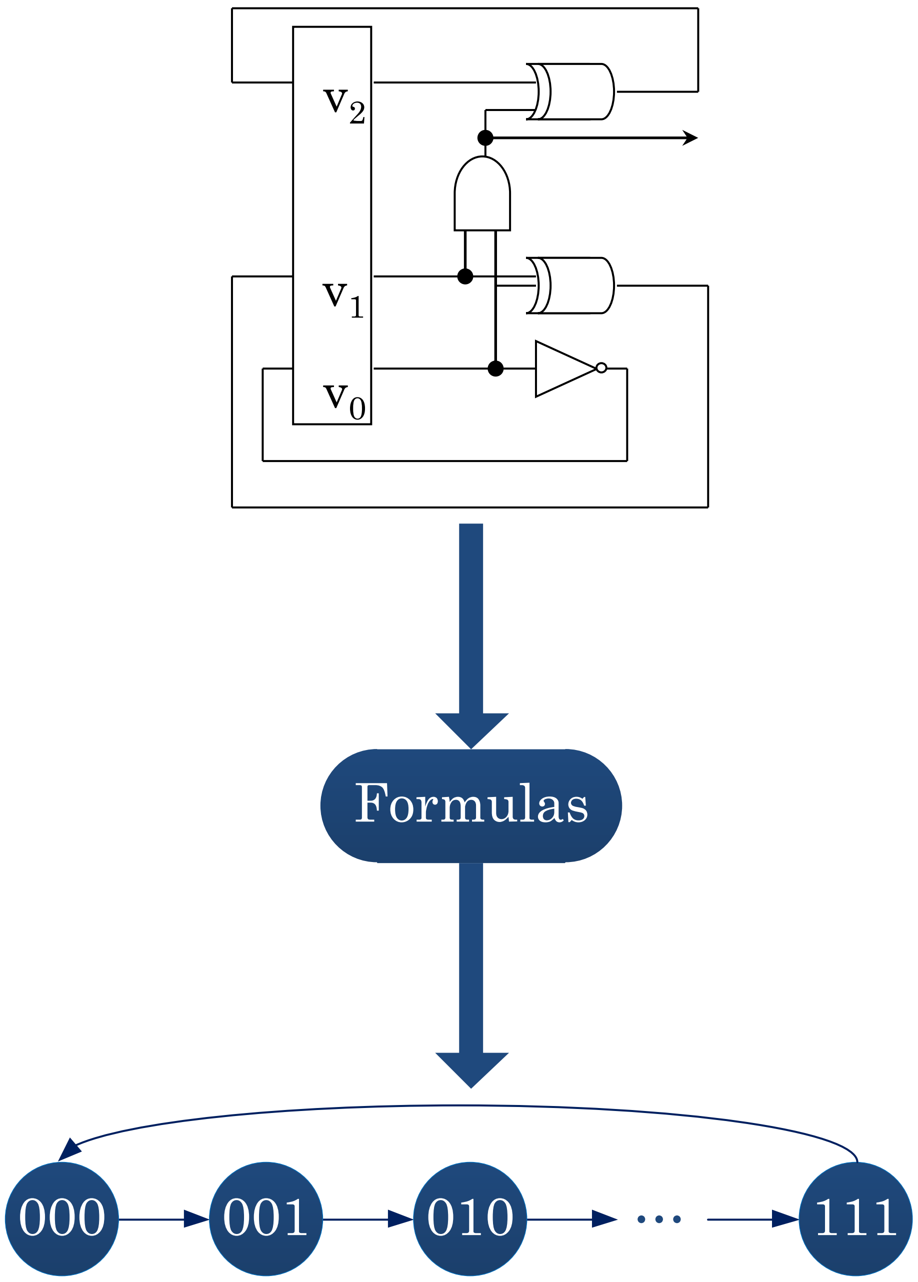


Concurrent systems

- A set of components that run simultaneously.
- Components can interact with each other.
  - *Execution*:
    - *asynchronous*
      - at any given time, only one component makes a calculation step
    - *synchronous*
      - all components take a calculation step at the same time
  - *Interaction*:
    - changes in the value of *shared variables*
    - *message exchange*
      - queues
      - handshaking

Digital circuits

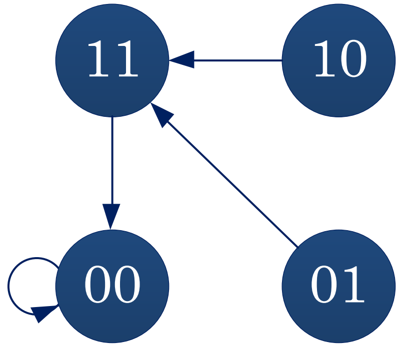
- Synchronous modulo 8 counter.
  - $V = \{v_0, v_1, v_2\}$ — state variables ,
  - $V` = \{v`_0, v`_1, v`_2\}$ — copy of the state variables.
  - Transitions:
    - $v`_0 = \neg v_0$
    - $v`_1 = v_0 \oplus v_1$
    - $v`_2 = (v_0 \wedge v_1) \oplus v_2$
  - Transition relations:
    - $R_0(V,V`) \equiv (v`_0 \Leftrightarrow \neg v_0)$
    - $R_1(V,V`) \equiv ( v`_1 \Leftrightarrow v_0 \oplus v_1)$
    - $R_2(V,V`) \equiv ( v`_2 \Leftrightarrow (v_0 \wedge v_1) \oplus v_2)$
  - All changes occur simultaneously:
    - $R(V,V`) \equiv R_0(V,V`) \wedge R_1(V,V`) \wedge R_2(V,V`)$.



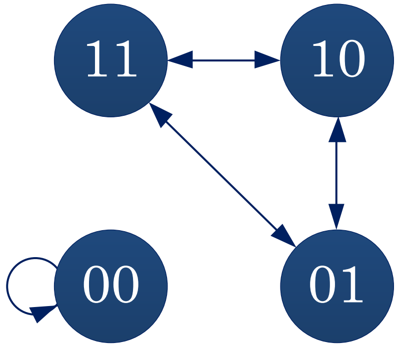
Digital circuits

- The difference between synchronous and asynchronous models
  - Let $V = \{v_0, v_1\}$, $v`_0 = v_0 \oplus v_1$ and $v`_1 = v_0 \oplus v_1$
  - Let s is the state with $v_0 = 1 \wedge v_1 = 1$

  - *Synchronous* model
    - One successor of state s
      - state $v_0 = 0 \wedge v_1 = 0$

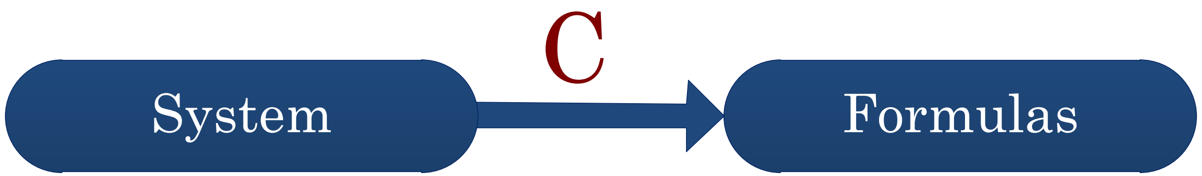


  - *Asynchronous* model
    - Two successors of state s:
      - $v_0 = 0 \wedge v_1 = 1$
      - $v_0 = 1 \wedge v_1 = 0$



Concurrent Programs

- Sequential programs
  - Consists of a sequence of statements
    - assignment, conditional statement, loop, etc.
- Concurrent programs
  - Combines sequential programs



- Translation *C* converts
  - text of program P into a first–order formula R, which represents the set of program transitions.
- Each operator has a single entry point and a single exit point.
  - These points are labeled in a unique way.
- For formulas of the system: define variables, initial states and transitions.
  - V — the set of program variables and program counters, and $V`$ — a copy of V.
- Initial states of the concurrent program P
  - $S_0(V,PC) = pre(V) \wedge pc = m \wedge \wedge^n_{i=1} (pc_i = \perp)$
    - $pc_i = \perp$ for process P_i is not active

System modeling: mutual exclusion

- P = m: **cobegin** $P_0 \parallel P_1$ **coend** m\`
- *pc* — program counter P
  - $pc = m$ — input label P
  - $pc = m$\` — output label P
  - $pc = \perp$, if P_0 and P_1 are active
- pc_i — program counter of process P_i
  - its values: l_i, $l`_i$, NC_i, CR_i , LC_i and $\perp$
- $V = V_0 = V_1 = \{$**turn**$\}$ and $PC = \{pc, pc_0, pc_1\}$.
- $pc_i = NC_i$: process i is in non–critical section.
  - A process waits (**turn** = i) to enter critical section.
- $pc_i = CR_i$: process i is in critical section.
  - It is forbidden to be in the critical section at the same time.
- $pc_i = LC_i$: process i goes out from critical section.
  - It changes **turn** for going out.

```
P0::    l0: while True do
    NC0:    wait (turn = 0);
    CR0:    actions;
    LC0:    turn := 1;
          od
        l`0

P1::    l1: while True do
    NC1:    wait (turn = 1);
    CR1:    actions;
    LC1:    turn := 0;
          od
        l`1
```

System modeling: mutual exclusion

- The *initial states* of concurrent program P:
  - $S_0(V, PC) \equiv pc = m \wedge pc_0 = \perp \wedge pc_1 = \perp$
    - There are no restrictions on the value of **turn**

- The formula for the *transition relation* $R(V, PC, V`, PC`)$ is
  - a disjunction of the formulas:
    - $pc = m \wedge pc`_0 = l_0 \wedge pc`_1 = l_1 \wedge pc` = \perp$
    - $pc_0 = l`_0 \wedge pc_1 = l`_1 \wedge pc` = m` \wedge pc`_0 = \perp \wedge pc`_1 = \perp$
    - $C(l_i, P_i, l`_i) \wedge same(V \setminus V_i) \wedge same(PC \setminus \{pc_i\})$
      - $C(l_i, P_i, l`_i) \wedge same(pc, pc_{\neg i}), i \in \{0,1\}$

```
P0::    l0: while True do
  NC0:      wait (turn = 0);
  CR0:      actions;
  LC0:      turn := 1;
          od
        l`0

P1::    l1: while True do
  NC1:      wait (turn = 1);
  CR1:      actions;
  LC1:      turn := 0;
          od
        l`1
```

System modeling: mutual exclusion

- For every P_i formula $C(l_i, P_i, l`_i)$ is disjunction:
  - $pc_i = l_i \wedge pc`_i = NC_i \wedge True \wedge same(\mathbf{turn})$
  - $pc_i = NC_i \wedge \mathbf{turn} \neq i \wedge same(pc_i, \mathbf{turn})$
  - $pc_i = NC_i \wedge pc`_i = CR_i \wedge \mathbf{turn} = i \wedge same(\mathbf{turn})$
  - $pc_i = CR_i \wedge \mathrm{C}(\mathbf{actions}) \wedge same(pc_i, \mathbf{turn})$
  - $pc_i = CR_i \wedge pc`_i = LC_i \wedge same(\mathbf{turn})$
  - $pc_i = LC_i \wedge pc`_i = l_i \wedge \mathbf{turn} = (i+1)(mod\ 2)$
  - $pc_i = l_i \wedge pc`_i = l`_i \wedge False \wedge same(\mathbf{turn})$

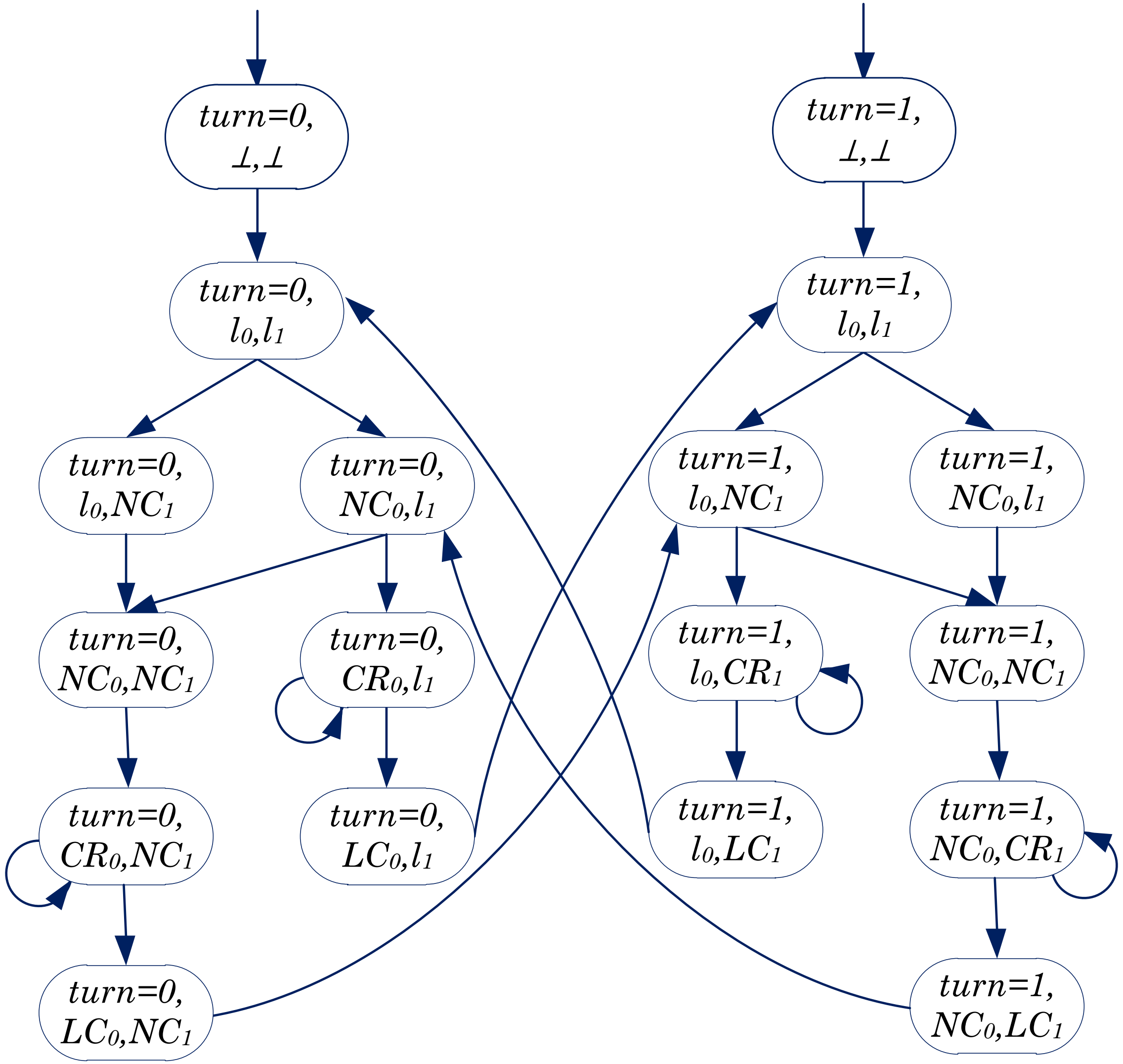
```
P0::     l0: while True do
      NC0:     wait (turn = 0);
      CR0:     actions;
      LC0:     turn := 1;
               od
         l`0

P1::     l1: while True do
      NC1:     wait (turn = 1);
      CR1:     actions;
      LC1:     turn := 0;
               od
         l`1
```

System modeling: mutual exclusion

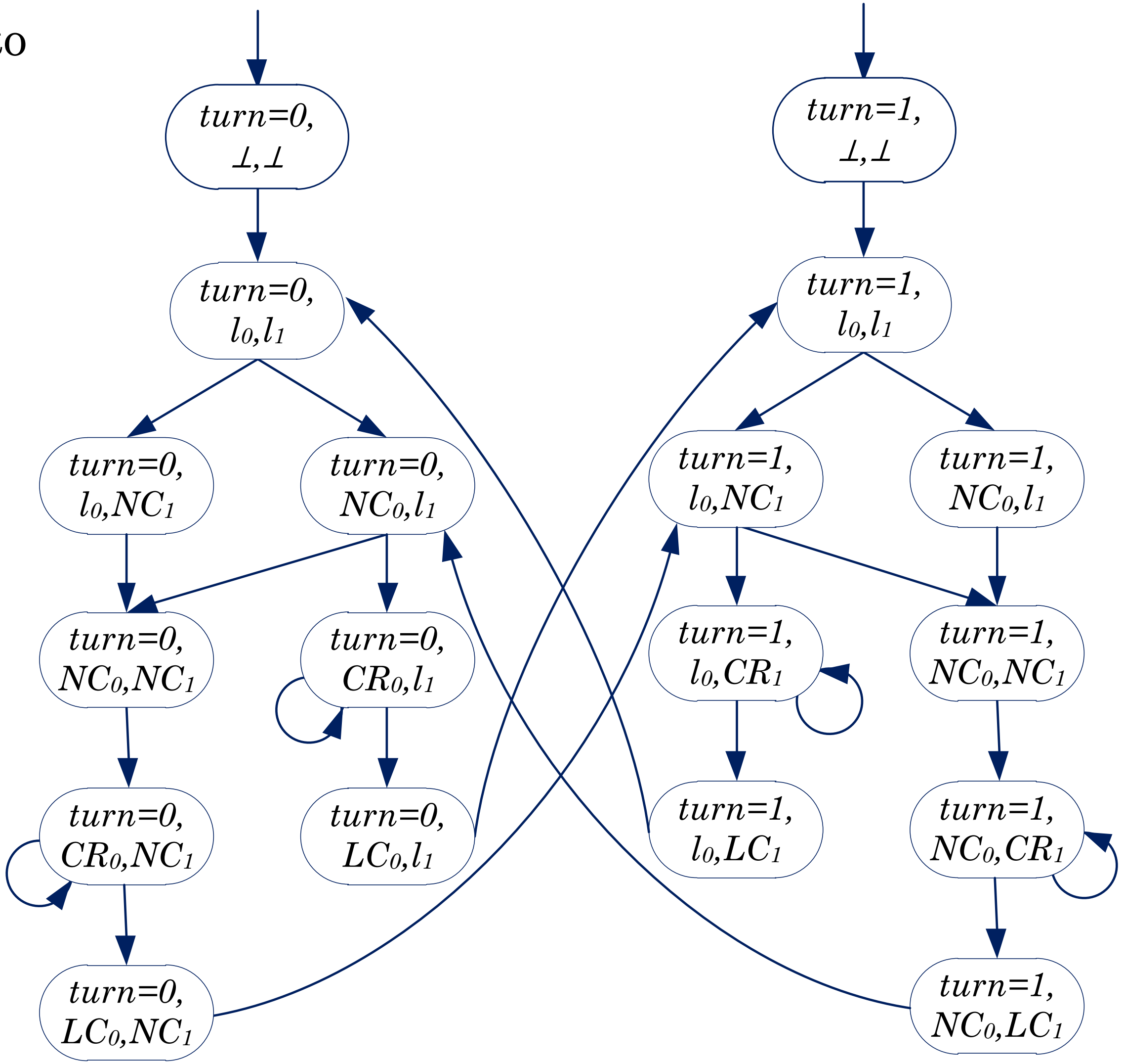
```
P0::    l0: while True do
        NC0:    wait (turn = 0);
        CR0:    actions;
        LC0:    turn := 1;
            od
        l`0

P1::    l1: while True do
        NC1:    wait (turn = 1);
        CR1:    actions;
        LC1:    turn := 0;
            od
        l`1
```



System modeling: mutual exclusion

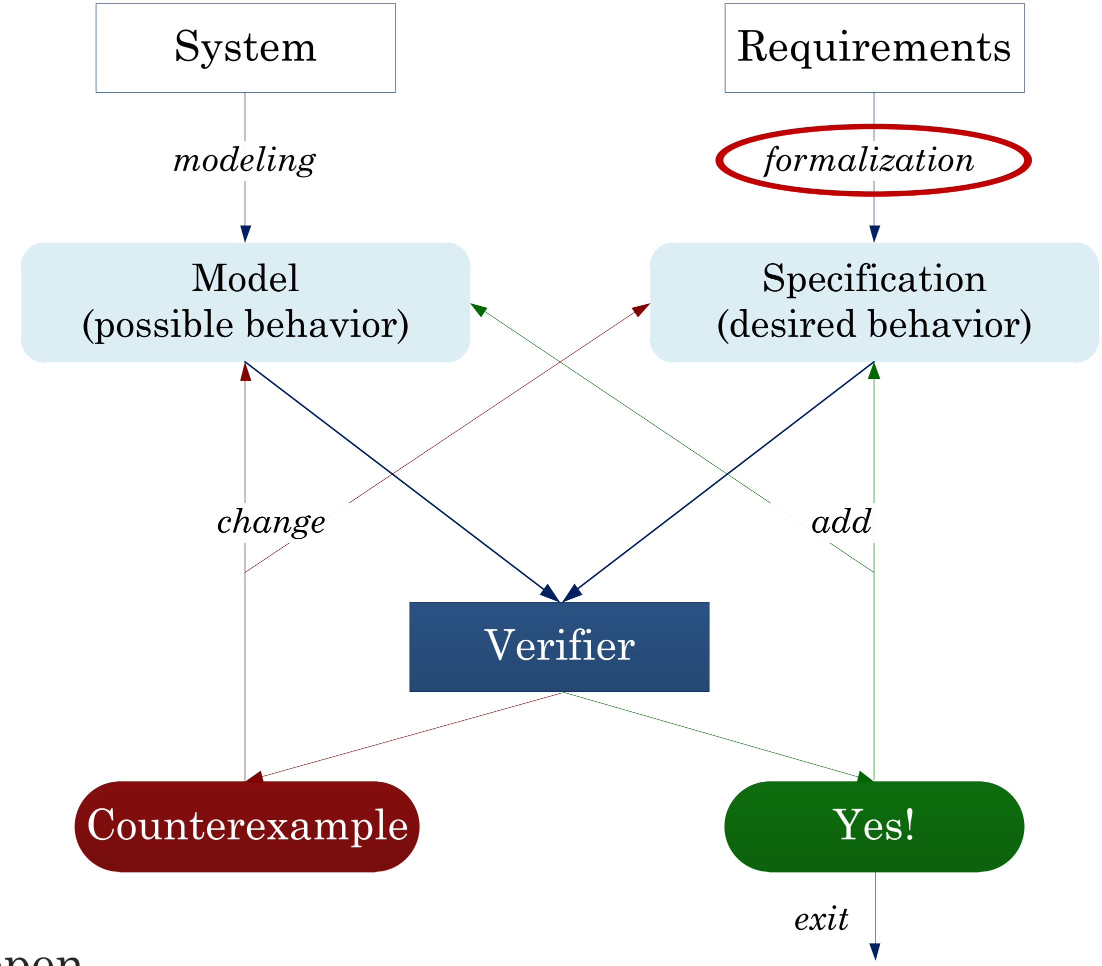
- The Kripke structure is constructed according to the formulas $\boldsymbol{S_0}$ and $\boldsymbol{R}$
  - *Mutual exclusion* property holds
    - Processes will never enter in their critical sections at the same time.
  - *Starvation* property does not hold
    - one process can continuously
      try to enter your critical section
      never getting access there,
      and at the same time
      another process is located
      in his critical section infinitely long.



Property Specification

Model checking process

- Modeling
  - Translating a description of
    a software system to the formal description
    suitable for automatic verification.
    - Compilation, translation.
    - Abstraction.
- Specification
  - Formulating software properties systems
    for verification
    - Logic: temporal, dynamic,
      epistemic, deontic and many others.
    - Completeness of the specification.
      - Safety: nothing bad will ever happen.
      - Liveliness: something good is definitely happen.
- Verification



Logics for specification: modal logics

- Temporal Logic 🕘
  - *LTL, CTL, CTL**
  - Statements about the evolution of a system over *time*
    - Something will surely happen sometime.
      - ***F** restart*
    - Something may always be.
      - ***EG** blocked*
    - There must have been something until something happened.
      - ***A** move **U** stop*
- Dynamic logic

Logics for specification

- Temporal Logic 🕒
- Dynamic Logic 💣
  - *PDL, PDL*, μ-calculus*
  - Statements about the evolution of a system as a result of certain *actions*.
    - If you do something, then you will definitely get something.
      - *[go_down] at_bottom*
    - If you do something repeatedly, then maybe you will get something.
      - *<go_up*> at_top*

Logics for specification

- Temporal Logic 🕓
- Dynamic Logic 💣
- Other logic
  - Epistemic 👓
    - *PLK, PLC*
    - Statements about *knowledge*.
      - I know that he knows that they are aware that I know a secret.
  - Deontic ☞
    - *SDL, DDL*
    - Statement about *obligations*
      - Something must take place.
  - Belief
  - Logic Combinations

Logics for specification

- Temporal Logic 🕙
- Dynamic Logic 💣
- Other logic
  - Epistemic 👓
  - Deontic ☞
  - Belief ☝
    - *PLB, PPLB*
      - Statements about *belief*
        - I believe something is true.
  - Combinations of logics 🕙 💣 👓 ☞ ☝
    - *All kinds* of statements
      - Someday, after some action, I find out that I always had to do something other before in order to believe that nothing would have changed due to I could know something obvious.

Linear Temporal Logic
LTL

Temporal logics

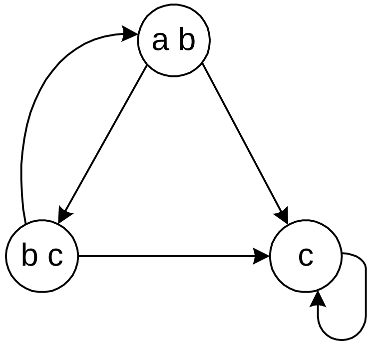
- Logics for the specification of temporal properties of transition systems (Kripke structures)
- Atomic propositions, Boolean connectives, quantifiers and modalities, operators describing the temporal properties of system states.
- Reactive systems
  - State transition sequence specification
    - *Floyd-Hoar method*
      - Input and Output Specification
    - *Temporal Logic*
      - Specification of the execution process (computation)

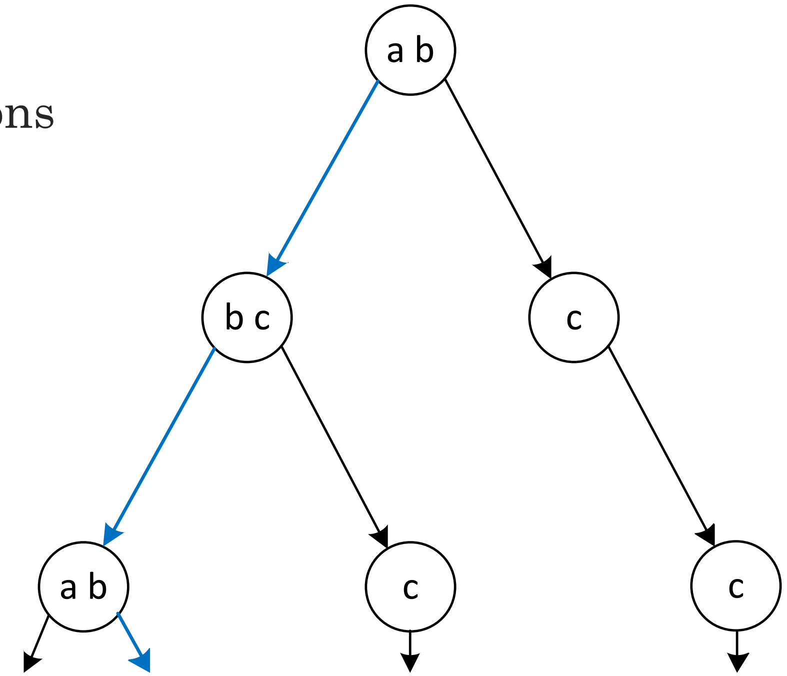
Temporal logics

- Time is not explicitly described:
  - in future
  - never
- Temporal operators for describing temporal properties.
- Temporal logics differ
  - the set of temporal operators used
  - semantics of these operators
- LTL – Linear Temporal Logic: about one path
- CTL – Computational Tree Logic: about alternating path
- MTL – Metric Temporal Logic: about measurable time on one path

Linear Temporal Logics LTL

- LTL formulas describe the properties of computation *paths*.
- *Computation tree*
  - Root — initial state
  - Successors — states obtained by transitions
  - Infinite
  - All possible computations





Linear Temporal Logics LTL

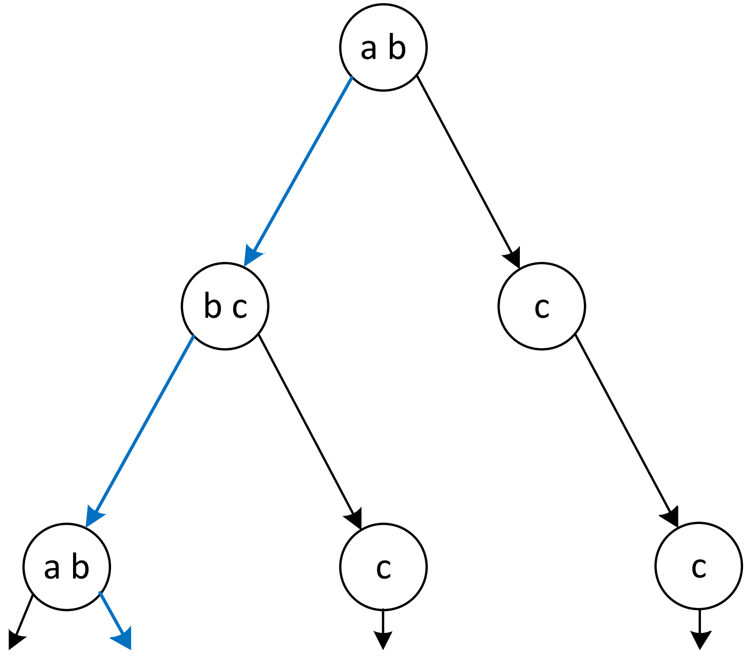
- Temporal operators
  - Properties of the path in a computation tree.
- Time shift operator ***X***
  - *ne**X**ttime*, "the next moment"
  - A property holds in the following state of the path.
- Incident operator ***F***
  - ***Future***, “sooner or later”, “sometime in the future”, “someday”
  - A property will hold in some subsequent state of the path.
- Invariance operator ***G***
  - ***Globally***, “always”, “everywhere”
  - A property holds in each state of the path.
- Conditional operator ***U***
  - ***Until***, “until”
  - A property holds starting from some point until some other property holds.

Syntax LTL formulas

An assertion about a path

$\neg\varphi,\ \varphi\wedge\psi,\ \varphi\vee\psi,$

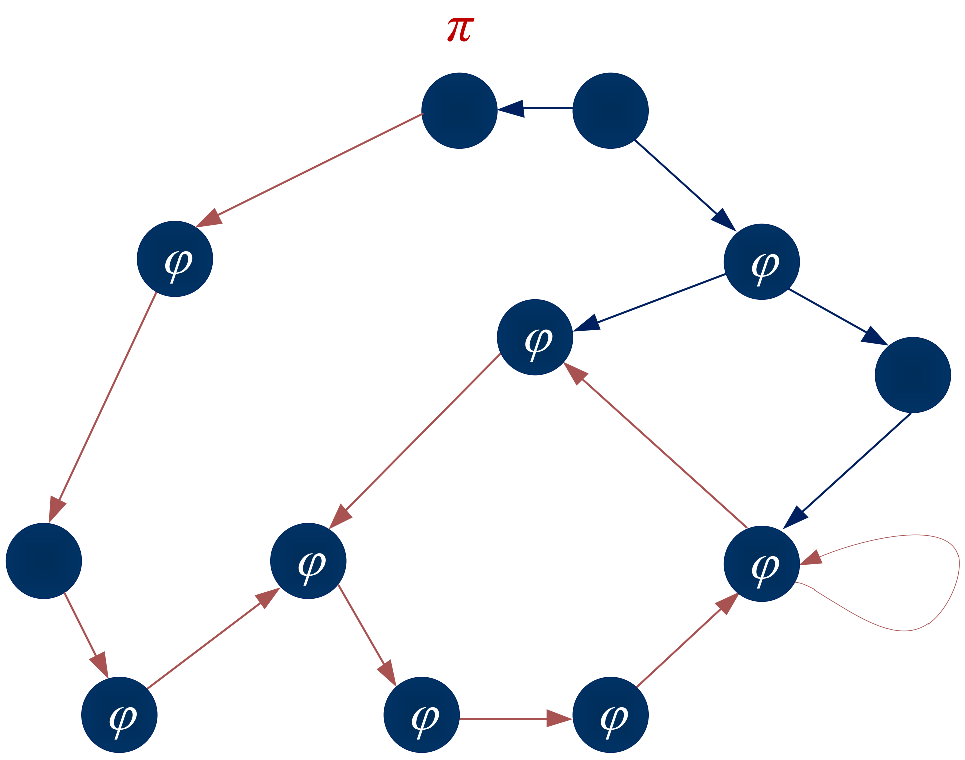
$\boldsymbol{X}\varphi,\ \boldsymbol{F}\varphi,\ \boldsymbol{G}\varphi,\ \varphi\boldsymbol{U}\psi$



Linear Temporal Logics LTL

Semantics of LTL formulas

- Based on Kripke structure $M = (S, S_0, R, L)$
  - S — finite set of states;
  - $S_0 \subseteq S$ — set of initial states;
  - $R \subseteq S \times S$ — total transition relation;
  - $L: S \rightarrow 2^{AP}$ — a labeling function.
- Path in model M from state s is
  - infinite sequence of model states
    $\pi = s_0, s_1, s_2, \dots$, such that $s_0 = s$ and for all $i \geq 0$ $R(s_i, s_{i+1})$.
    - infinite branch in the computation tree of model M
- $M, \pi \models \varphi$ — φ holds on path π of model M



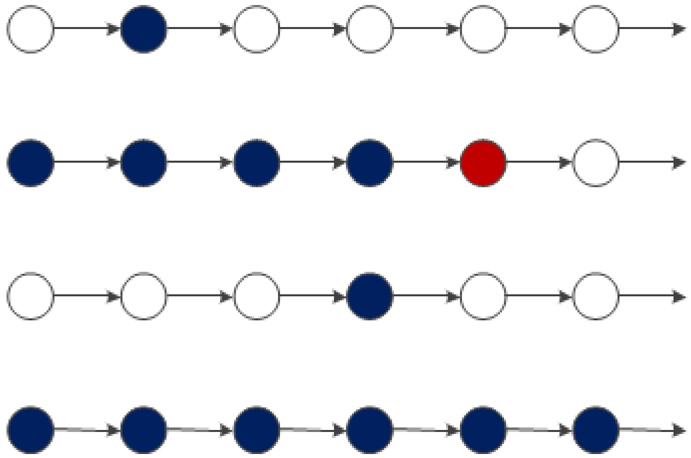
Linear Temporal Logics LTL

- Relation $\models$ is defined inductively
  - φ, φ_1 and φ_2 are formulas
  - path $\pi = s, s_1, \ldots s_k, \ldots$ and suffix $\pi^k = s_k, \ldots s_m, \ldots$
- $M, \pi \models p \quad \Leftrightarrow \quad p \in L(s)$
- $M, \pi \models \neg\varphi \quad \Leftrightarrow \quad M, \pi \not\models \varphi$
- $M, \pi \models \varphi_1 \wedge \varphi_2 \quad \Leftrightarrow \quad M, \pi \models \varphi_1$ *and* $M, \pi \models \varphi_2$
- $M, \pi \models \varphi_1 \vee \varphi_2 \quad \Leftrightarrow \quad M, \pi \models \varphi_1$ or $M, \pi \models \varphi_2$

Linear Temporal Logics LTL

- Relation $\models$ is defined inductively
  - φ, φ_1 and φ_2 are formulas
  - path $\pi = s, s_1, \ldots s_k, \ldots$ and suffix $\pi^k = s_k, \ldots s_m, \ldots$
- $M, \pi \models \boldsymbol{X}\varphi \quad \Leftrightarrow \quad M, \pi^1 \models \varphi$
- $M, \pi \models \varphi_1 \; \boldsymbol{U} \; \varphi_2 \quad \Leftrightarrow$ there exists $k \geq 0$, such that

  $\pi^k \models \varphi_2$ and for all $0 \leq j < k$: $M, \pi^j \models \varphi_1$ holds
- $M, \pi \models \boldsymbol{F}\varphi \quad \Leftrightarrow$ there exists $k \geq 0$ such that $M, \pi^k \models \varphi$
- $M, \pi \models \boldsymbol{G}\varphi \quad \Leftrightarrow$ for all $k \geq 0$: $M, \pi^k \models \varphi$ holds



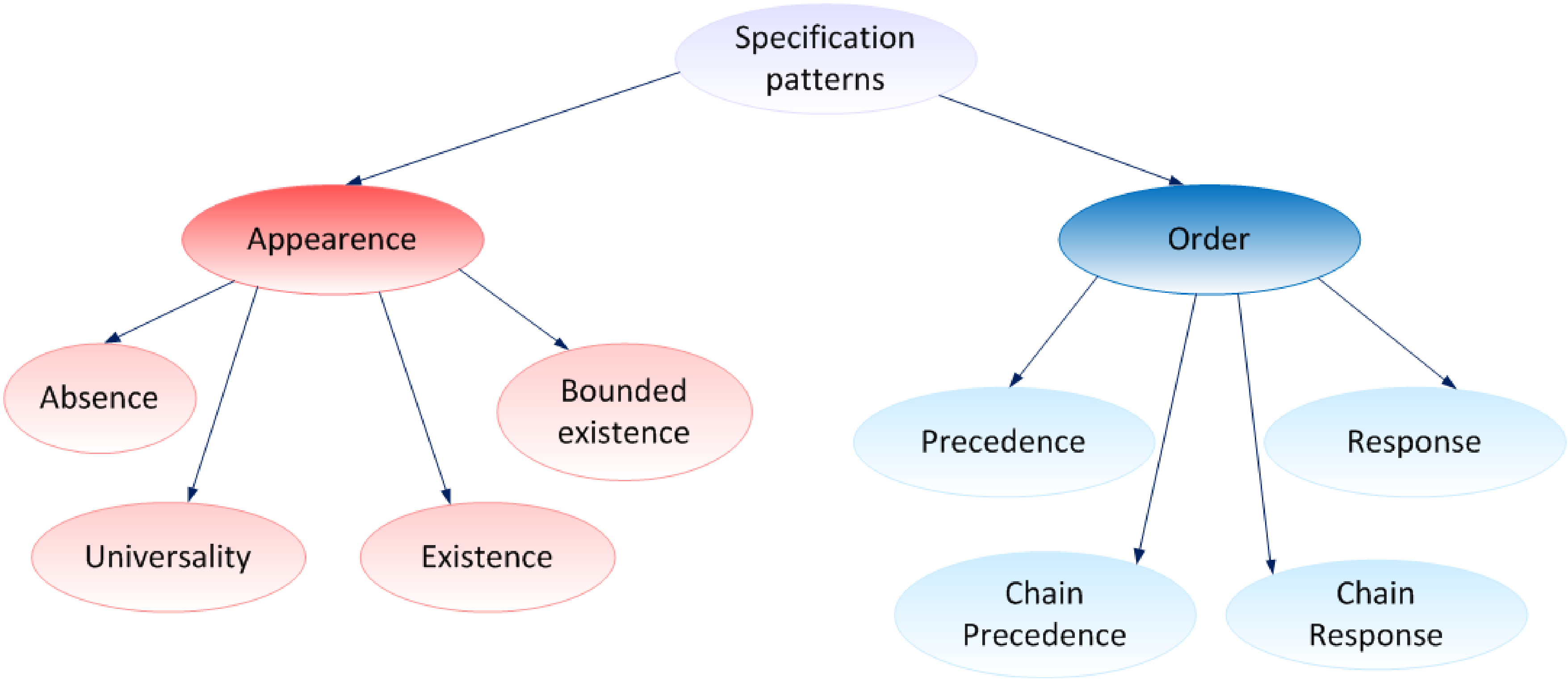
$$\boldsymbol{F}\varphi \equiv true \; \boldsymbol{U}\varphi$$
$$\boldsymbol{G}\varphi \equiv \neg \boldsymbol{F} \neg \varphi$$

Typical LTL formulas

- $\boldsymbol{F}\ (Start \wedge \neg Ready)$
  - it is possible to achieve a state in which the *Start* condition is satisfied, and *Ready* is not.
- $\boldsymbol{G}\ (Req \rightarrow \boldsymbol{F}\ Ack)$
  - if a request is received, it will be confirmed sooner or later.
- $\boldsymbol{GF}\ (DeviceEnabled)$
  - *DeviceEnabled* condition holds infinitely often on every system execution.
- $\boldsymbol{GF}\ (Restart)$
  - from any state, *Restart* state is reachable.
- $\boldsymbol{GF}\varphi$
  - formula φ holds infinitely often (*liveness*)
- $\boldsymbol{FG}\varphi$
  - eventually φ becomes true and holds forever (*stabilization*)

CTL and LTL: specification patterns

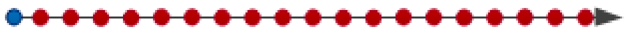
- https://matthewbdwyer.github.io/psp/patterns.html



Specification patterns

Occurrence patterns

- appearance of a system event *p* during system execution.

- Universality:
  - *Meaning*: it is always the case that *p* holds
  - *Semantics* LTL: **G**p
  - *Graphical form*: 
  - *Example*: Alice loves Bob **forever**. 

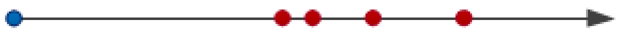
X *– next*
F *– future*
G *– always*
U *– until*
W *– weak until*

Specification patterns

Occurrence patterns

- appearance of a system event *p* during system execution.

- Existence:
  - *Meaning*: *p* eventually holds
  - *Semantics* LTL: **F**p
  - *Graphical form*:



  - *Example*: Alice will love Bob **sometime**.

X – *next*
F – *future*
G – *always*
U – *until*
W – *weak until*

Specification patterns

Occurrence patterns

- appearance of a system event *p* during system execution.

- Absence:
  - *Meaning*: it is never the case that *p* holds
  - *Semantics* LTL: $\mathbf{G}\ \neg p$
  - *Graphical form*:

  - *Example*: Alice and Bob are **never** friends.

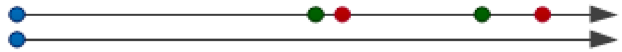
X – *next*
F – *future*
G – *always*
U – *until*
W – *weak until*

Specification patterns

Order patterns

- a relative appearance of events p and s during execution.

- Precedence:
  - *Meaning*: if p happens, then s has happened before
  - *Semantics* LTL: $\neg p \ \mathbf{W} \ (s \wedge \neg p \wedge \mathbf{F} p)$
  - *Graphical form*:



  - *Example*: The *cooling* is only possible after *overheating*.

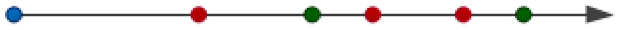
X – *next*
F – *future*
G – *always*
U – *until*
W – *weak until*

Specification patterns

Order patterns

- a relative appearance of events p and s during execution.

- Response:
  - *Meaning*: it is always the case that if p holds, then s eventually holds
  - *Semantics* LTL: $\mathbf{G}(p \rightarrow \mathbf{F}s)$
  - *Graphical form*:



  - *Example*: After the *overheating signal* the *cooling* will start.

X – *next*
F – *future*
G – *always*
U – *until*
W – *weak until*

Specification patterns: eventual restrictions

- Property ***p***, property ***r***, property ***q***

- Globally
- Before ***r***
- After ***q***
- Between ***q*** and ***r***
- After ***q***, until ***r***.

Specification patterns: eventual restrictions

Pattern – *what*, scope – *when*.

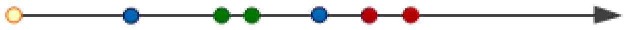
- Globally: no restrictions.
- Before R: a pattern holds during program execution before *r* first occurs.
  - Universality Before R:
    - *Meaning*: *p* always holds until *r* holds
    - *Semantics* LTL: $\mathbf{F}r \rightarrow p\mathbf{U}r$
    - *Graphical form*:

    - *Example*: Alice and Bob are friends **until** *Carlos* comes.

X – *next*
F – *future*
G – *always*
U – *until*
W – *weak until*

Specification patterns: eventual restrictions

Pattern – *what*, scope – *when*.

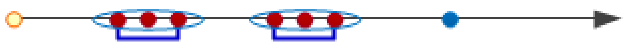
- Before R: a pattern holds during program execution before r first occurs.
  - Response Before R:
    - *Meaning*: if p holds, then s eventually holds before r holds
    - *Semantics* LTL: $\mathbf{F}r \rightarrow (p \rightarrow (\neg r\ \mathbf{U}(s \wedge \neg r)))\mathbf{U}r$
    - *Graphical form*: 
    - *Example*: After the *overheating signal* the *cooling* starts before *the temperature becomes low*.

X *– next*
F *– future*
G *– always*
U *– until*
W *– weak until*

Restrictions: eventual, durational, quantitative

- Globally + Before + Duration + Quantity:
  - *Meaning*: p happens 2 two times before r and its duration is k time unites.
  - *Formal semantics* MTL:

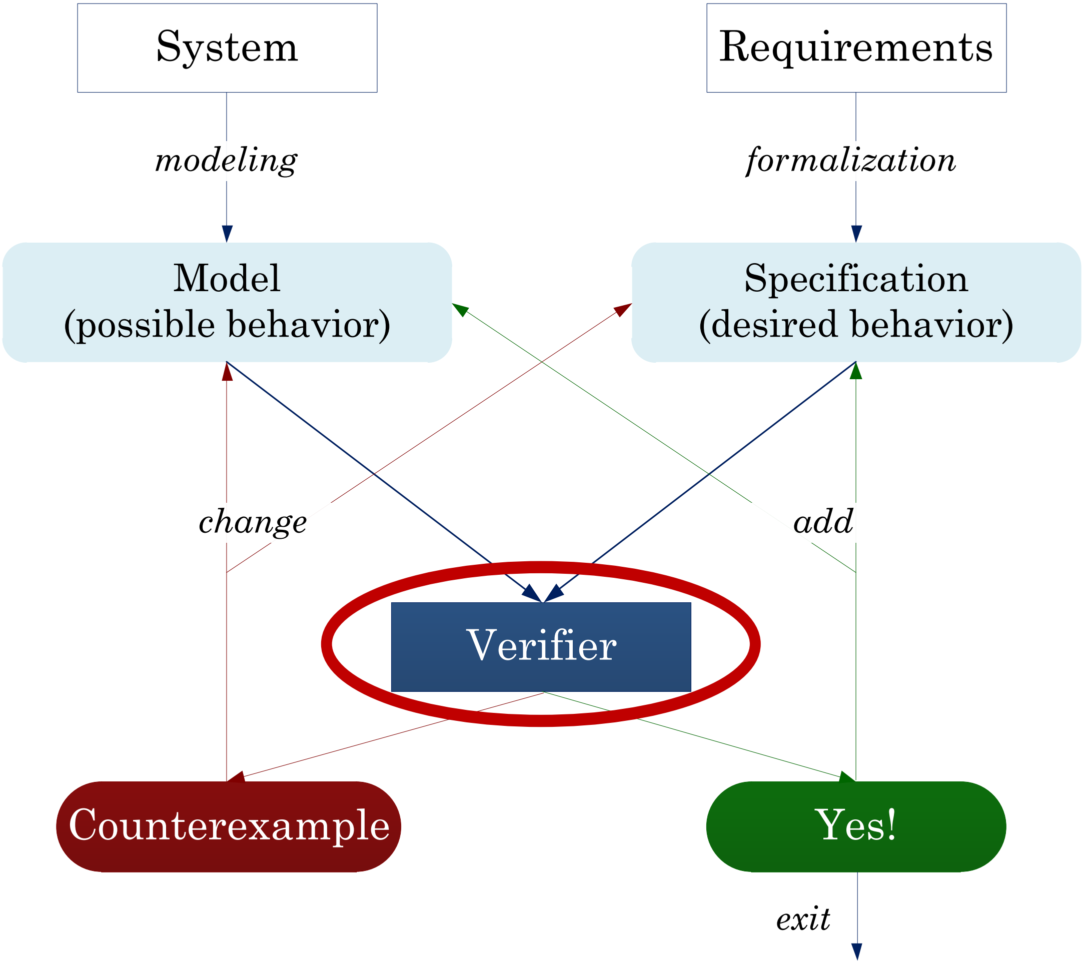
$$\mathbf{F}r \rightarrow (\neg r\mathbf{U}(\mathbf{F}\mathbf{G}_k(p\wedge\neg r) \wedge \mathbf{X}(\neg r\mathbf{U}(\mathbf{F}\mathbf{G}_k(p\wedge\neg r) \wedge \mathbf{X}(\neg\mathbf{G}_k p\mathbf{U}\ r)))))$$

  - *Graphical form*: 
  - *Example*: A train must give two long calls before train maintenance staff release the brakes.

LTL model checking

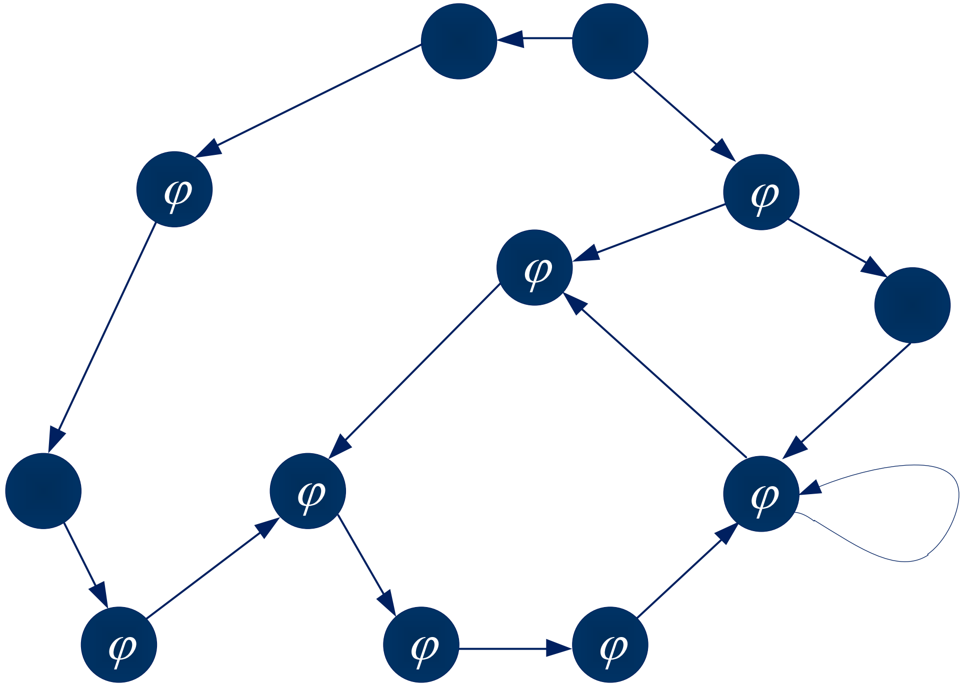
Model checking process

- Modeling
- Specification
- Verification
  - Checking consistency of
    the model and its specification.
    - Fully automatic.
    - Results Analysis
      - Counterexample
      - False counterexample
        - Model refinement
    - Large size model.
      - Abstraction



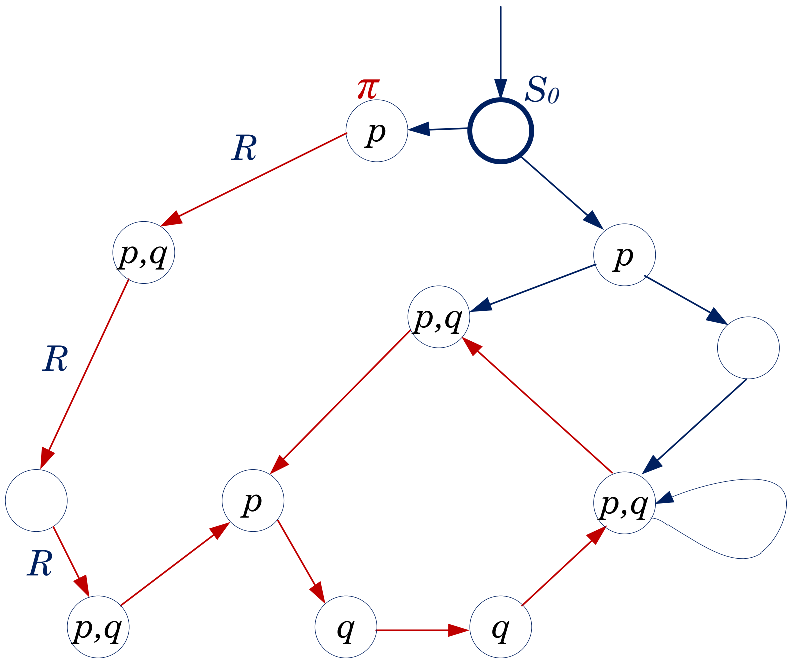
Model checking problem

- Model checking problem
  - For given
    - Kripke structure $M = (S, S_0, R, L)$
      - finite state system
    - formula of (temporal) logic φ
      - specification
  - To find in the set S
    - the set of all states in which φ holds, i.e.
      - *semantics of φ*, the set $\{s \in S \mid M, s \models \varphi\}$.
  - If a concurrent system has initial states, then
    - the *system satisfies the specification* φ if
      - all the initial states in the semantics of φ.



Model checking problem

- The abstract algorithms for solving the model checking problem
  - explicit representation of Kripke structures in the form of labeled directed graphs
    - vertices — states from S,
    - edges — transition relation R,
    - vertex labeling — function $L : S \to 2^{AP}$.



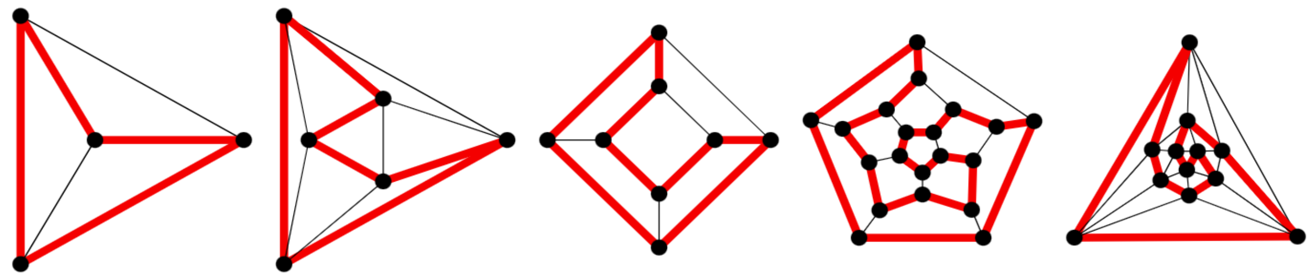
LTL Model Checking

- Kripke structure $M = (S, S_0, R, L)$, $s \in S$, $\boldsymbol{A}\varphi \in ALTL$.
  - $\neg\varphi$, $\varphi\wedge\psi$, $\varphi\vee\psi$, $\boldsymbol{X}\varphi$, $\boldsymbol{F}\varphi$, $\boldsymbol{G}\varphi$, $\varphi\boldsymbol{U}\psi$ in LTL

- Model checking problem
  - Verify that $M,s \models \boldsymbol{A}\varphi$
    - $\Leftrightarrow$ $\mathrm{M,s} \models \neg\mathbf{E}\neg\varphi$
      - it is enough to be able to check $\boldsymbol{E}\varphi$.

- Model checking problem for $\boldsymbol{E}\varphi$ is *NP-complete*.
  - Let $G = (V, A)$ is graph with $V = \{v_1, \dots, v_n\}$.
  - The *Hamiltonian path* finding problem can be reduced to checking if $M, s \models \varphi$

$$\boldsymbol{E}[\boldsymbol{F}p_1 \wedge \dots \wedge \boldsymbol{F}p_n \wedge \boldsymbol{G}(p_1 \rightarrow \boldsymbol{XG}\neg p_1) \wedge \dots \wedge \boldsymbol{G}(p_n \rightarrow \boldsymbol{XG}\neg p_n)]$$



Tableaux LTL Model Checking

- Tableaux algorithm of Liechtenstein-Pnueli
  - The complexity is
    - exponential with respect to the length of the verifying formula and
    - linear with respect to the size of the model
- *Tableaux* is graph constructed according to the formula
  - The algorithm for checking the satisfiability of LTL formula in Kripke structure
    - constructs the composition of the tableaux and structure to verify if
      - there is a *computation* in the structure that simultaneously
      - is the *path* in the tableaux.
- The model checking problem for LTL-formula and the Kripke structure is to determine
  - whether a given formula holds on all paths
    - or if there are paths on which the formula does not hold.

Tableaux LTL Model Checking

- For the given formula φ and the Kripke structure M,
  - construct the tableaux T for the formula φ.
    - The Kripke structure which includes just all the paths that satisfy formula φ.

Tableaux LTL Model Checking

$\boldsymbol{X}$ – *next*
$\boldsymbol{F}$ – *future*
$\boldsymbol{G}$ – *always*
$\boldsymbol{U}$ – *until*
$\boldsymbol{W}$ – *weak until*

Transform the formula to normal form – negations before propositions only.

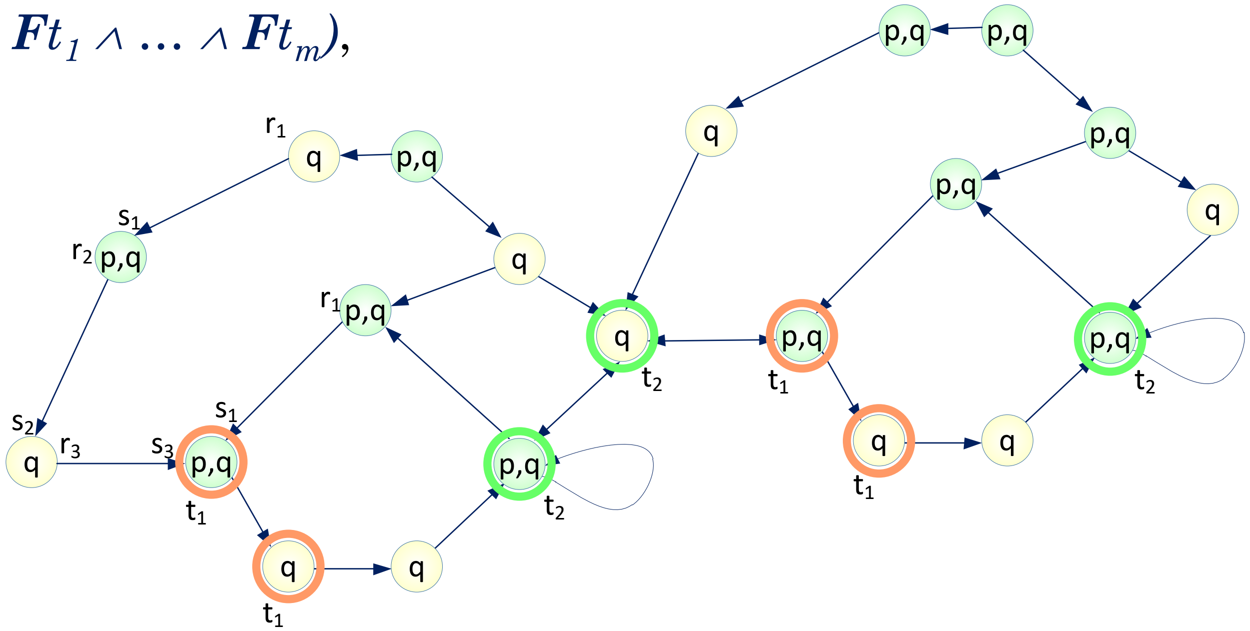
- $\neg\neg\varphi \equiv \varphi$
- $\neg(\varphi \wedge \psi) \equiv \neg\varphi \vee \neg\psi$
- $\neg(\varphi \vee \psi) \equiv \neg\varphi \wedge \neg\psi$
- $\varphi \rightarrow \psi \equiv \neg\varphi \vee \psi$
- $\neg\boldsymbol{X}\varphi \equiv \boldsymbol{X}\neg\varphi$
- $\neg\boldsymbol{F}\varphi \equiv \boldsymbol{G}\neg\varphi$
- $\neg\boldsymbol{G}\varphi \equiv \boldsymbol{F}\neg\varphi$
- $\neg(\varphi\, \boldsymbol{U}\, \psi) \equiv \neg\psi\, \boldsymbol{W}\, \neg(\varphi \vee \psi)$
- $\neg(\varphi\, \boldsymbol{W}\, \psi) \equiv \neg\psi\, \boldsymbol{U}\, \neg(\varphi \vee \psi)$
- The algorithm also works for general formulas.

X – *next*
F – *future*
G – *always*
U – *until*
W – *weak until*

Towards a tableaux form for LTL–formula

<u>Transforming LTL formulas to tableaux form</u>

- $p \wedge \mathbf{G}(q \wedge (r_1 \rightarrow \mathbf{X}s_1) \wedge \ldots \wedge (r_n \rightarrow \mathbf{X}s_n) \wedge \mathbf{F}t_1 \wedge \ldots \wedge \mathbf{F}t_m)$,
- p – the initial conditions
- q – the invariant
- $r_i \rightarrow \mathbf{X}s_i$ – the transitions
- t_j – the fairness conditions



- Two steps:
  1. Coding temporal formulas with atomic propositions
  2. Replacing temporal operators

X – *next*
F – *future*
G – *always*
U – *until*
W – *weak until*

Towards a tableaux form for LTL–formula

1. Coding temporal formulas

- The formula f_S is obtained from the formula f by replacing the subformulas f_i containing temporal operators with atomic statements p_i, starting with the deepest.
- Formula $f_C = \bigwedge_i \mathbf{G}(p_i \to f_i)$.
  - Always, if proposition p_i holds then formula f_i holds also.
    - Proposition p_i «codes» f_i.
- $f \equiv f_S \wedge f_C$
- $f = \neg a \vee \mathbf{G}\neg b \wedge \mathbf{X}(\neg c\ \mathbf{U}\ (\neg d \wedge \neg c))$
  - $f_S = \neg a \vee p_1 \wedge p_3$
  - $f_C = \mathbf{G}(p_1 \to \mathbf{G}\neg b) \wedge \mathbf{G}(p_3 \to \mathbf{X}p_2) \wedge \mathbf{G}(p_2 \to (\neg c\ \mathbf{U}\ (\neg d \wedge \neg c)))$

$$p \wedge \mathbf{G}(q \wedge (r_1 \to \mathbf{X}s_1) \wedge \ldots \wedge (r_n \to \mathbf{X}s_n) \wedge \mathbf{F}t_1 \wedge \ldots \wedge \mathbf{F}t_m)$$

X – *next*
F – *future*
G – *always*
U – *until*
W – *weak until*

Towards a tableaux form for LTL–formula

2. Replacing temporal operators

- $f \equiv f_S \wedge f_C = F \wedge \bigwedge_i \mathbf{G}(p_i \to f_i)$
  - F do not include temporal operators
  - $h_i = \bigwedge_i \mathbf{G}(p_i \to f_i)$
- Replace h_i including $\mathbf{G}$, $\mathbf{F}$, $\mathbf{U}$ and $\mathbf{W}$.

1. $h_i = \mathbf{G}(p_i \to \mathbf{G}d_i) \equiv \mathbf{G}(p_i \to r_i) \wedge \mathbf{G}(r_i \to d_i) \wedge \mathbf{G}(r_i \to \mathbf{X}r_i)$
   - new r_i promises that d_i will hold always.
2. $h_i = \mathbf{G}(p_i \to \mathbf{F}d_i) \equiv \mathbf{G}(p_i \to (\neg d_i \to r_i)) \wedge \mathbf{G}(r_i \to \mathbf{X}(\neg d_i \to r_i)) \wedge \mathbf{GF}\, \neg r_i$
3. $h_i = \mathbf{G}(p_i \to (d_i\, \mathbf{W}\, e_i)) \equiv \mathbf{G}(p_i \to e_i \vee d_i \wedge r_i) \wedge \mathbf{G}(r_i \to \mathbf{X}(e_i \vee d_i \wedge r_i))$
4. $h_i = \mathbf{G}(p_i \to (d_i\, \mathbf{U}\, e_i)) \equiv \mathbf{G}(p_i \to e_i \vee d_i \wedge r_i) \wedge \mathbf{G}(r_i \to \mathbf{X}(e_i \vee d_i \wedge r_i)) \wedge \mathbf{G}(p_i \to \mathbf{F}e_i)$

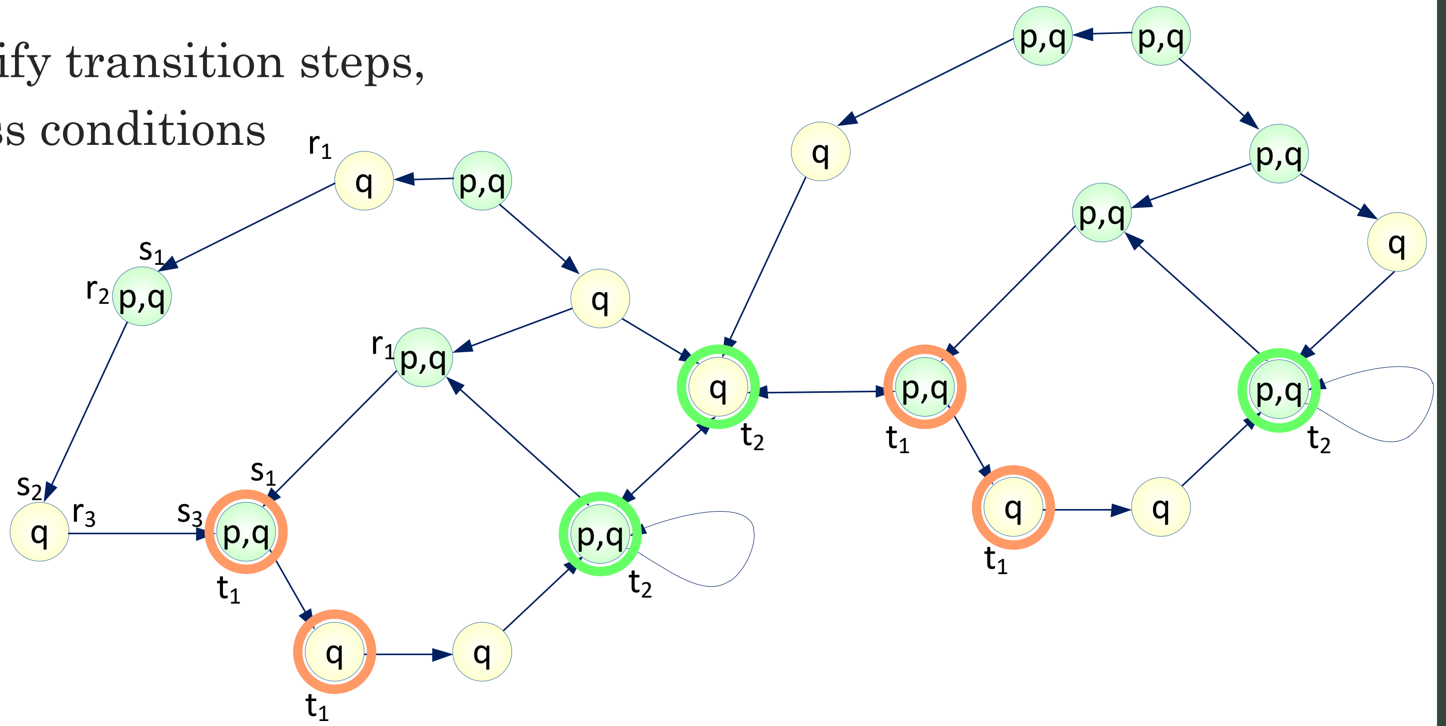
$$p \wedge \mathbf{G}(q \wedge (r_1 \to \mathbf{X}s_1) \wedge \ldots \wedge (r_n \to \mathbf{X}s_n) \wedge \mathbf{F}t_1 \wedge \ldots \wedge \mathbf{F}t_m)$$

A tableaux form for LTL–formula

- Factor out all operators $\boldsymbol{G}$ and propositional subformulas:

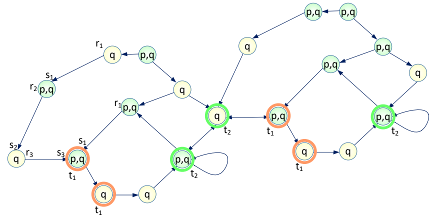
$$p \wedge \boldsymbol{G}(q \wedge (r_1 \to \boldsymbol{X}s_1) \wedge \ldots \wedge (r_n \to \boldsymbol{X}s_n) \wedge \boldsymbol{F}t_1 \wedge \ldots \wedge \boldsymbol{F}t_m),$$

  - $p, q, r_1, \ldots, r_n, s_1, \ldots, s_n, t_1, \ldots, t_m$ — propositional subformulas without temporal operators.
  - formula p «plays just at the start»,
  - formula q «plays always»,
  - formulas r_i and s_i (i in $[1..n]$) specify transition steps,
  - formulas t_j (j in $[1..m]$) are fairness conditions reachable from every state.



Tableaux LTL Model Checking

- Tableaux form $\varphi \equiv p \wedge \boldsymbol{G}(q \wedge (r_1 \rightarrow \boldsymbol{X}s_1) \wedge \ldots \wedge (r_n \rightarrow \boldsymbol{X}s_n) \wedge \boldsymbol{F}t_1 \wedge \ldots \wedge \boldsymbol{F}t_m)$
- Tableaux T_φ – the graph of semantics of φ



Model Checking

- Does LTL formula φ hold in a given structure M?
- The language of this structure $L_\omega(M)$ is the set of (infinite) paths of M.
- Naive verification method:
  - Is $L_\omega(M) \subseteq L_\omega(T_\varphi)$ true?
  - Accepting paths in structure M – *possible* system behavior
  - Accepting table paths T_φ – *desired* system behavior
  - If any *possible* behavior is the *desired*: $L_\omega(M) \subseteq L_\omega(T_\varphi)$
    - then the model satisfies the formula φ
  - The problem of inclusion checking for languages of Kripke structures is *PSPACE*-complete.

Tableaux LTL Model Checking

$$L_\omega(M) \subseteq L_\omega(T_\varphi) \quad \text{iff} \quad L_\omega(M) \cap L_\omega(\neg T_\varphi) = \varnothing,$$

- $\neg T_\varphi$ is complement of T_φ
  - A structure with the set of paths which are complement of the set of paths in T_φ.
- The best known algorithm for constructing $\neg T_\varphi$ by tableaux T_φ
  - is quadratically exponential:
    - if T_φ includes n states, then $\neg T_\varphi$ includes $c^{n \hat{} 2}$ states for some $c > 1$.

Tableaux LTL Model Checking

$$L_\omega(M) \subseteq L_\omega(T_\varphi) \quad \text{iff} \quad L_\omega(M) \cap L_\omega(\neg T_\varphi) = \varnothing,$$

- Compliment structure for $\neg T_\varphi$ is equivalent to structure for $\neg\varphi$:
  - $L_\omega(\neg T_\varphi) = L_\omega(T_{\neg\varphi})$

Model checking algorithm:

1. Build a tableaux for the negation of the desired property φ.
   - Tableaux $T_{\neg\varphi}$ models undesirable system computations

If M contains an acceptable path, which is also an acceptable path in $T_{\neg\varphi}$

- an example of a computation that violates the property φ, i.e. φ is not a property of M.
- Otherwise, φ holds.

2. Check if $L_\omega(M) \cap L_\omega(T_{\neg\varphi}) = \varnothing$.
   - Construct a Cartesian product of the structures M and $T_{\neg\varphi}$.

Tableaux LTL Model Checking

- Structure $M = (S, S_0, R, L)$, structure $T_{\neg\varphi} = (S_D, S_{0D}, R_D, L_D)$
- Product structure $M' = (S', R', S_0', L')$:
- $S' = \{(s, s_D) \mid s \in S, s_D \in S_D \text{ and } L_D(s_D) \cap AP = L(s)\}$;
- $R' = \{((s, s_D), (s', s_D')) \mid (s, s') \in R, (s_D, s_D') \in R_D\} \cap (S' \times S')$;
- $S_0' = \{(s_0, s_{0D}) \mid s_0 \in S_0, s_{0D} \in S_{0D}\} \cap S'$;
- $L(s, s_D) = L_D(s_D)$.

3. Determine whether in the model M' there exists an infinite path starting from the initial state and satisfying the fairness conditions t_j
   - If such a path exists, then in M there exists a path satisfying $\neg\varphi$.
     - A *counterexample* for the formula φ.

- The time complexity of this algorithm $O(|M| \times 2^{|\varphi|})$.

Summary

- Model Checking (formal verification)
  - Exhaustive search in parallel system states
    - States, transitions
      - Kripke structure
        - Formal model for programs and systems in general
          - Translation
  - Checks if a specification is satisfiable in a state
    - Specification logics
      - Temporal logics
        - Something happens in time
      - Other logics