Formal Methods in Software Development

Introduction to Temporal Logic(s)
Linear Time Properties

Ivano Salvo

Computer Science Department

SAPIENZA
UNIVERSITÀ DI ROMA

Lesson **2**, October 12th, 2020

Lesson 2a

Defining Specifications

Kripke structures

For verification purposes, we usually **drop action labels:** there are useful mainly for **synchronizations** purposes

Let AP be a set of atomic proposition. A **Kripke structure** $\mathcal{M}$ over AP is a 4-tuple (S, S_0, R, L), where:

- S is a finite set of **states**;
- $S_0 \subseteq S$ is the set of **initial states**;
- $R \subseteq S \times S$ is the **transition relation**;
- $L : S \rightarrow 2^{AP}$ is the **labeling function**.

R must be **total**, i.e. for each state s there exists always s' such that $R(s, s')$

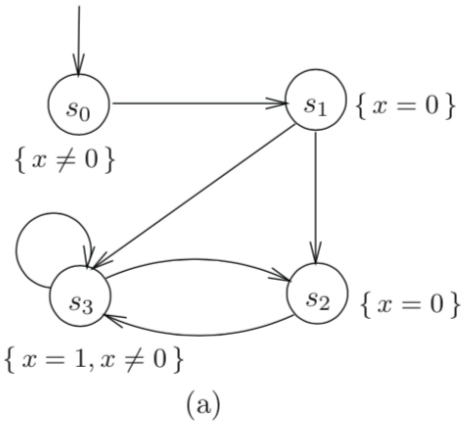
A **path** (or **execution**) from s in $\mathcal{M}$ is a sequence $\pi = s_0 s_1 s_2 \ldots$ such that $s_0 = s$ and $R(s_i, s_{i+1})$

Temporal Logic

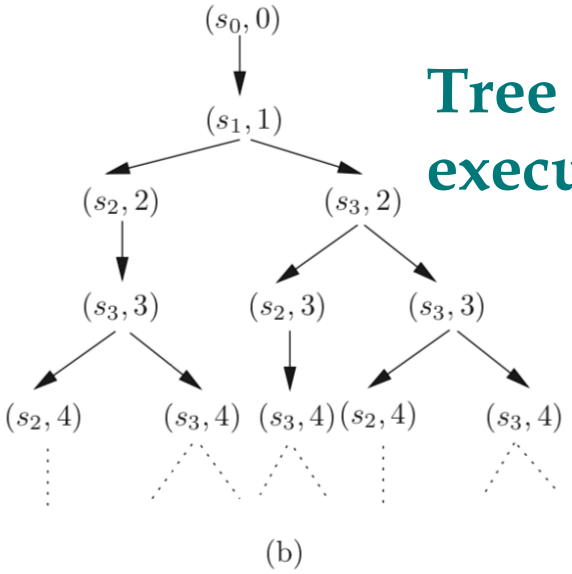
First order logic is useful to describe properties of sequential programs. Reactive/concurrent systems interact with their environment, and hence their **sequences of computation** (and its properties) are of primary importance.

Temporal Logic focuses on sequence of transitions, or better on **the tree** of possible (**usually infinite**) **executions** of a system.

Temporal operators: *never*, in the *future*, *always*, *eventually*.



Kripke structure



Tree of executions

*Computational Tree Logic: CTL**

CTL* formulas are built starting from **atomic propositions** and **propositional connectives** ($\wedge$, $\vee$, $\neg$, $\rightarrow$ etc.)

Path quantifiers: **A** (**for all** computation paths, **aka** $\forall$) and **E** (**for some** computation path, **aka** $\exists$). They quantify over paths starting in a given state (**state formulas**)

Temporal operators (originates **path formulas**):

X *f* ("**next time**", **aka** $\circ$) *f* holds in the second state of a path

F *f* ("**eventually**" or "**in the future**", **aka** $\Diamond$) *f* will hold at some state on the path

G *f* ("**globally**", **aka** $\Box$) *f* holds at every state on the path

f **U** *g* ("**until**") combines two properties. *f* **U** *g* holds if *g* holds at some state on the path and *f* holds until that point.

F *g* ≡ true **U** *g*

R ("**release**") is the dual of **U**. *f* **R** *g* holds if *g* holds up to a state where *f* holds (***g* may hold forever and *f* never!**).

CTL*: syntax

State formulas are formulas that depend on a state of a transition system

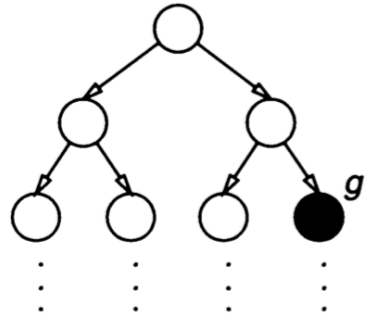
- If $p \in AP$, then p is a **state formula**
- If f, g are **state formulas**, then $\neg f$, $f \wedge g$, $f \vee g$ are **state formulas**
- If f is a **path formula**, the $\mathbf{A}\, f$ and $\mathbf{E}\, f$ are **state formulas**

Path formulas are formulas that depend on a computation path

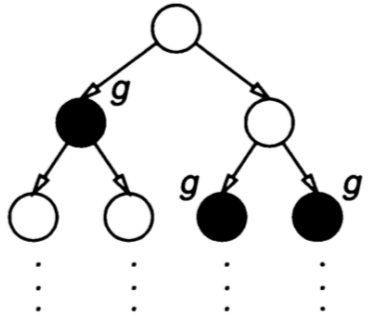
- If p is a **state formula**, then p is also a **path formula**
- If f, g are **path formulas**, then $\neg f$, $f \wedge g$, $f \vee g$, $\mathbf{X}\, f$, $\mathbf{F}\, f$, $\mathbf{G}\, f$, $f\, \mathbf{U}\, g$, and $f\, \mathbf{R}\, g$ are **path formulas**

Examples

The semantics of CTL* formulas are relative to a **computation tree**.

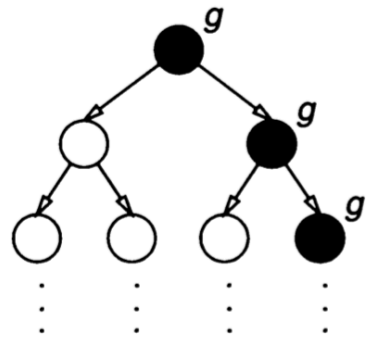


$M, s_0 \models \mathbf{EF}\, g$

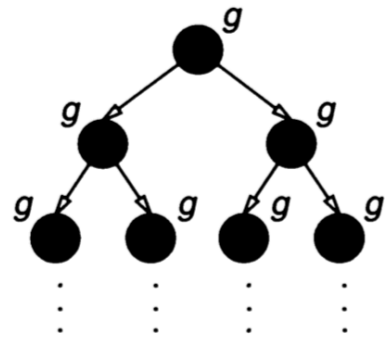


$M, s_0 \models \mathbf{AF}\, g$

Here some example of computation trees and CTL* formulas valid in such computation trees.



$M, s_0 \models \mathbf{EG}\, g$



$M, s_0 \models \mathbf{AG}\, g$

CTL* semantics: state formulas

The truth of a CTL* **state formula** is given in terms of a state s in a Kripke structure $\mathcal{M}$, notation $\mathcal{M}, s \vDash f$

1.	$M, s \models p$	$\Leftrightarrow$	$p \in L(s)$.
2.	$M, s \models \neg f_1$	$\Leftrightarrow$	$M, s \not\models f_1$.
3.	$M, s \models f_1 \vee f_2$	$\Leftrightarrow$	$M, s \models f_1$ or $M, s \models f_2$.
4.	$M, s \models f_1 \wedge f_2$	$\Leftrightarrow$	$M, s \models f_1$ and $M, s \models f_2$.
5.	$M, s \models \mathbf{E}\ g_1$	$\Leftrightarrow$	there is a path π from s such that $M, \pi \models g_1$.
6.	$M, s \models \mathbf{A}\ g_1$	$\Leftrightarrow$	for every path π starting from s, $M, \pi \models g_1$.

CTL* semantics: path formulas

The truth of a CTL* **path formula** is given in terms of a path π in a Kripke structure $\mathcal{M}$, notation $\mathcal{M}, \pi \models f$

Notation: π^i (or $\pi[i..]$) denotes the suffix of π starting in s_i

7.	$M, \pi \models f_1$	$\Leftrightarrow$	s is the first state of π and $M, s \models f_1$.
8.	$M, \pi \models \neg g_1$	$\Leftrightarrow$	$M, \pi \not\models g_1$.
9.	$M, \pi \models g_1 \vee g_2$	$\Leftrightarrow$	$M, \pi \models g_1$ or $M, \pi \models g_2$.
10.	$M, \pi \models g_1 \wedge g_2$	$\Leftrightarrow$	$M, \pi \models g_1$ and $M, \pi \models g_2$.
11.	$M, \pi \models \mathbf{X}\, g_1$	$\Leftrightarrow$	$M, \pi^1 \models g_1$.
12.	$M, \pi \models \mathbf{F}\, g_1$	$\Leftrightarrow$	there exists a $k \geq 0$ such that $M, \pi^k \models g_1$.
13.	$M, \pi \models \mathbf{G}\, g_1$	$\Leftrightarrow$	for all $i \geq 0$, $M, \pi^i \models g_1$.
14.	$M, \pi \models g_1\, \mathbf{U}\, g_2$	$\Leftrightarrow$	there exists a $k \geq 0$ such that $M, \pi^k \models g_2$ and for all $0 \leq j < k$, $M, \pi^j \models g_1$.
15.	$M, \pi \models g_1\, \mathbf{R}\, g_2$	$\Leftrightarrow$	for all $j \geq 0$, if for every $i < j$ $M, \pi^i \not\models g_1$ then $M, \pi^j \models g_2$.

Minimal CTL fragment*

It is easy to see that (for example) operators $\vee$, $\neg$, **X**, **U**, and **E** are enough to define formulas equivalent to any CTL* formulas via **duality**.

$$f \wedge g \equiv \neg(\neg f \vee \neg g)$$

$$f \mathbf{R} g \equiv \neg(\neg f \mathbf{U} \neg g)$$

$$\mathbf{F} f \equiv \text{true } \mathbf{U} f$$

$$\mathbf{G} f \equiv \text{false } \mathbf{R} f$$

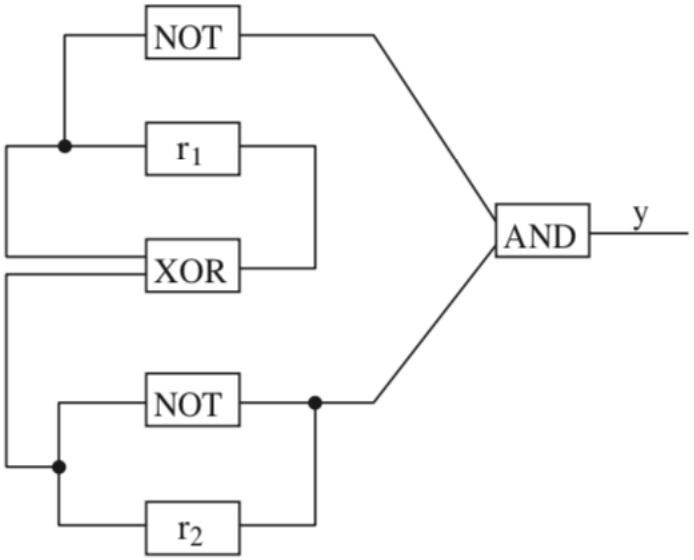
$$\mathbf{G} f \equiv \neg \mathbf{F} \neg f$$

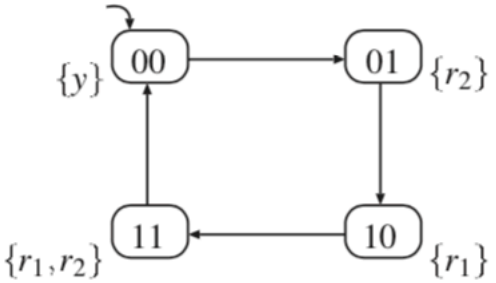
$$\mathbf{A} f \equiv \neg \mathbf{E} \neg f$$

In the following, we analyze two important **sub-logic of CTL***:

- **Linear Time Logic** (**LTL**)
- **Computational Tree Logic** (**CTL**)

Example of neXt: modulo 4 counter





This system satisfies the property:

$$G\,(\,y \rightarrow (\mathbf{X}\,\neg y \wedge \mathbf{X}\,\mathbf{X}\,\neg y \wedge \mathbf{X}\,\mathbf{X}\,\mathbf{X}\,\neg y))$$

that means that ***y* holds exactly every four steps** (forever)

Lesson 2b:

Linear Time Logic
LTL

Linear Time Logic (LTL)

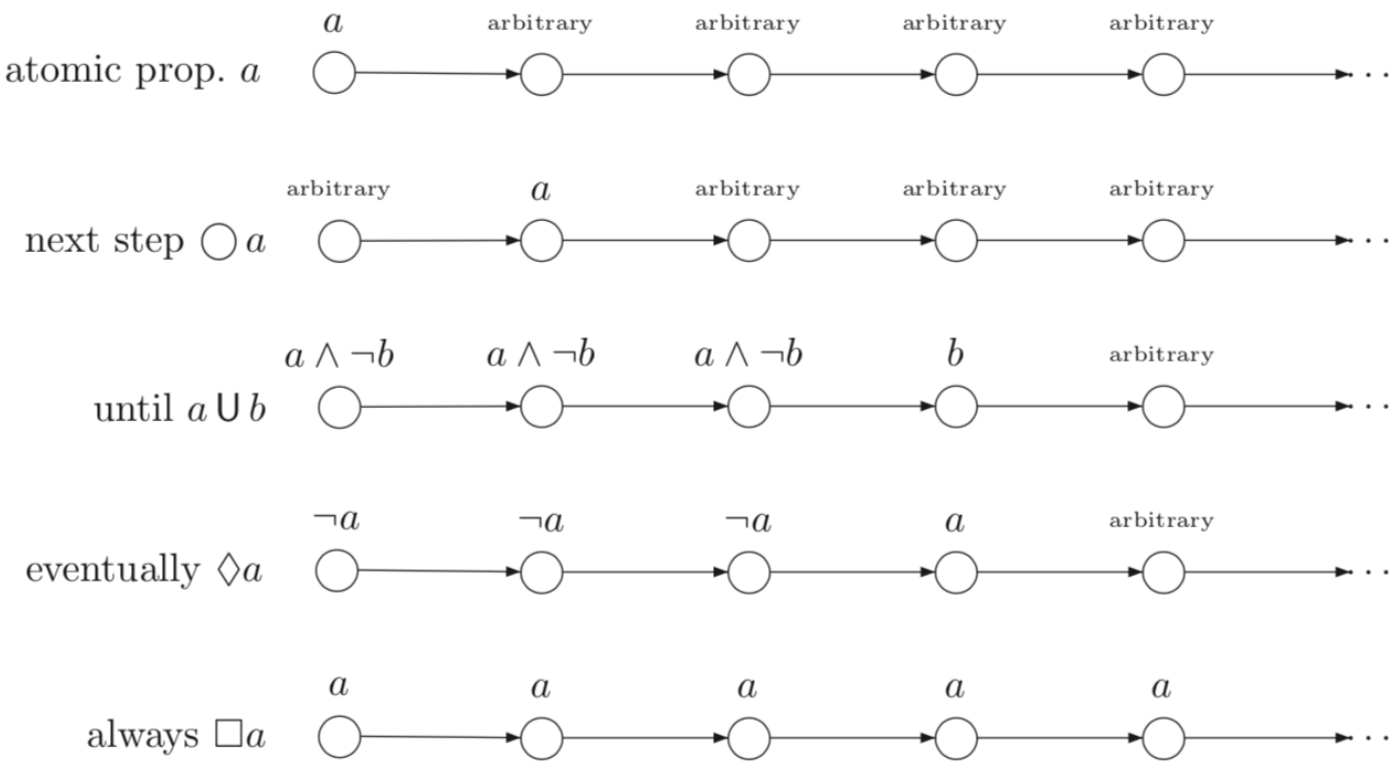
LTL is a fragment of CTL* where formulas have the form **A** *f* with *f* a path formula. Differently from CTL* path formulas are just built from **atomic propositions** using **temporal operators** (**no nested occurrences** of **A** or **E**)

- If $p \in AP$, then p is also a path formula
- If f, g are path formulas, then $\neg f$, $f \wedge g$, $f \vee g$, **X** f, **F** f, **G** f, f **U** g, and f **R** g are path formulas

To a LTL formula φ, it is associated a **LT property**, defined by **the set of paths π such that $\pi \models \varphi$** (see semantics of CTL* -- LTL is a sublogic of **path formulas**)

$$\mathcal{M} \models \varphi \Leftrightarrow \text{for all } s \in S_0,\ \mathcal{M}, s \models \varphi$$

LTL: Semantics



Some useful algebraic laws

Expansion Laws:

$$f \mathbf{U} g \equiv g \vee (f \wedge \mathbf{X} (f \mathbf{U} g))$$
$$\mathbf{F} f = f \vee \mathbf{X} \mathbf{F} f$$
$$\mathbf{G} f = f \wedge \mathbf{X} \mathbf{G} f$$

These are crucial in LTL model checking algorithm: recursive definition of words that satisfies such formulas.

Idempotency Laws:

$$\mathbf{F} \mathbf{F} f \equiv \mathbf{F} f$$
$$\mathbf{G} \mathbf{G} f \equiv \mathbf{G} f$$
$$f \mathbf{U} (f \mathbf{U} g) \equiv f \mathbf{U} g$$
$$(f \mathbf{U} g) \mathbf{U} g \equiv f \mathbf{U} g$$

Absorption Laws:

$$\mathbf{G} \mathbf{F} \mathbf{G} f \equiv \mathbf{F} \mathbf{G} f$$
$$\mathbf{F} \mathbf{G} \mathbf{F} f \equiv \mathbf{G} \mathbf{F} f$$

Lesson 2c

Linear Properties

Linear Time Properties

Linear time properties depend on traces (**system executions**)

Safety properties: something **bad never happens**

- Deadlock
- Invariants (state properties, eg. mutual exclusion)
- Trace properties (e.g. beverage is delivered only after the coin has been inserted)

Liveness properties: something **good will eventually happen**

- starvation freedom (the process will eventually enter in the critical section)
- some event will happen **infinitely often**.

Liveness and **safety** properties are **dual** and both needed to specify a reasonable system.

Example: systems that do nothing are for sure safe! But probably useless!

Traces and LT properties

Traces are infinite **words of sets of atomic propositions**.
Atomic propositions is **what we observe** of a system state.

$$\text{traces}(\mathcal{M}) = \bigcup_{s \in S_0} \text{traces}(\mathcal{M}, s) \subseteq (2^{AP})^{\omega}$$

Traces can be easily obtained by execution paths of a LTS, by **dropping action names** and **substituting each state *s* with its labeling *L*(*s*)** [the same for Kripke structures].

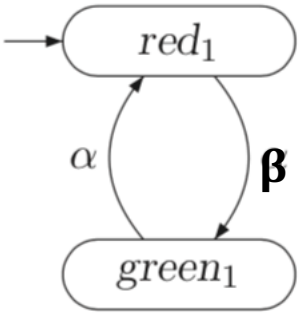
A **Linear Time property** P is just a subset of $(2^{AP})^{\omega}$

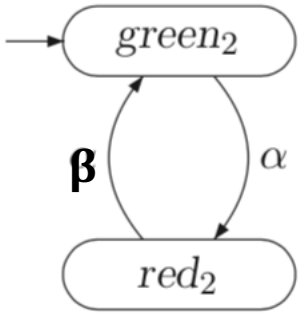
$$\mathcal{M} \models P \quad \text{if and only if} \quad \text{traces}(\mathcal{M}) \subseteq P$$

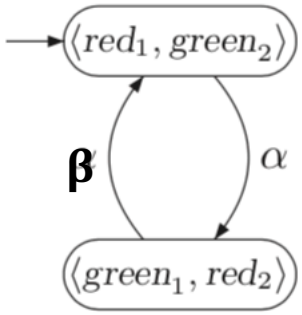
Observation: For convenience, $\mathcal{M}$ is without terminal states, therefore we reason about **infinite words**.

Remember: execution paths start in **initial states**.

Example: traffic lights: properties







Two traffic lights and they parallel composition via **handshaking**.

P_S = "The traffic lights are never both green simultaneously"
= **(A) G** ¬ ($green_1 \wedge green_2$)

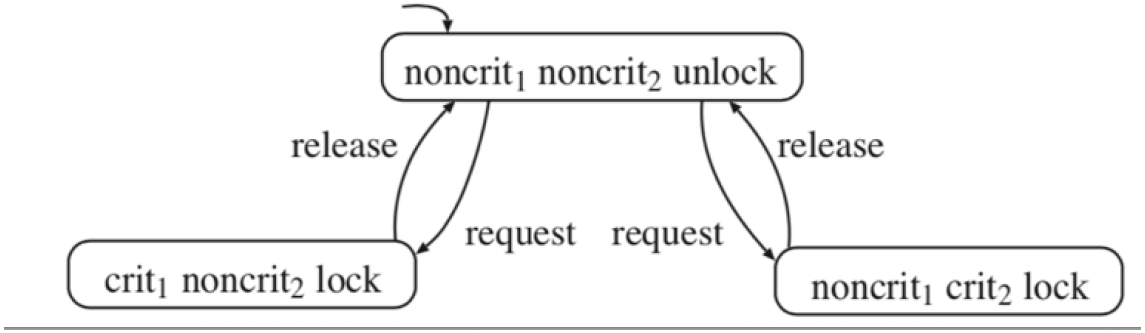
safety

P_L = "The first traffic light will be green infinitely often"
= **(A) G F** $green_1$

liveness

Both P_L and P_S **are satisfied** by this system, since traces have the form {red_1, $green_2$} {red_2, $green_1$} {red_1, $green_2$} {red_2, $green_1$}…

Mutual Exclusion: handshaking



Does this system satisfy the mutual exclusion property?

$$\mathbf{G}\,(\neg\, crit_1 \vee \neg\, crit_2) \equiv \mathbf{G}\,\neg(\, crit_1 \wedge crit_2) \equiv \neg\, \mathbf{F}\,(\, crit_1 \wedge crit_2)$$

Does it satisfy the following liveness properties?

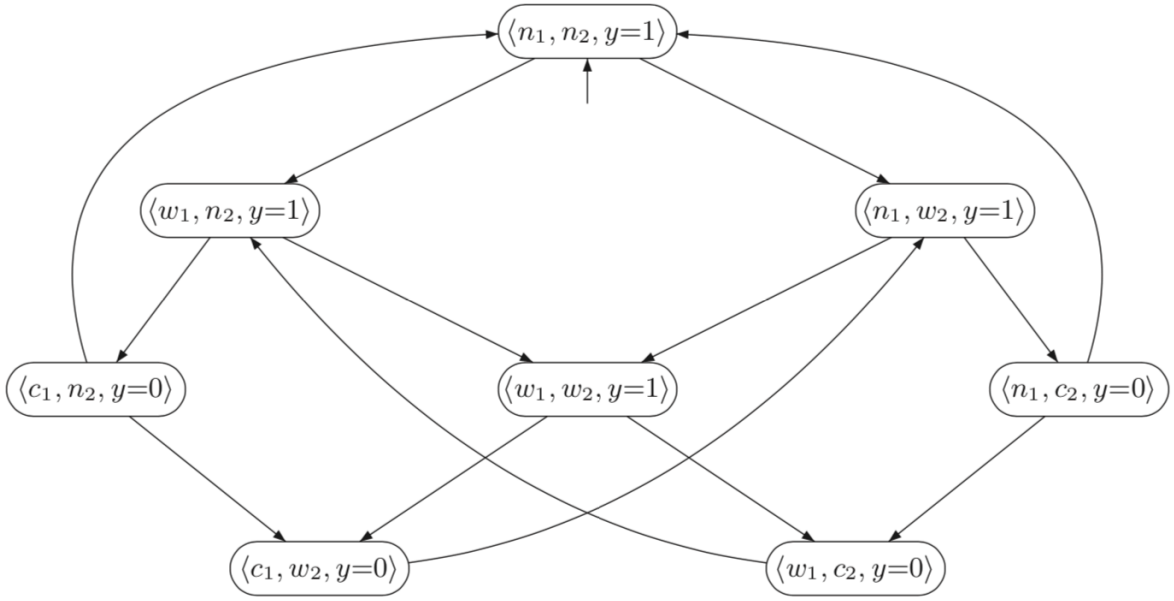
* each process enters in its critical section:

$$(\mathbf{F}\ crit_1) \wedge (\mathbf{F}\ crit_2)$$

* each process enters infinitely often in the critical section)

$$(\mathbf{G}\ \mathbf{F}\ crit_1) \wedge (\mathbf{G}\ \mathbf{F}\ crit_2)$$

Mutual Exclusion via semaphores



This system satisfies:

$$\mathbf{G}\ (y = 0 \Longrightarrow crit_1 \vee crit_2)$$

but again, no liveness, even in weaken forms, such as:

$$(\mathbf{G}\ \mathbf{F}\ wait_1 \Longrightarrow \mathbf{G}\ \mathbf{F}\ crit_1) \wedge (\mathbf{G}\ \mathbf{F}\ wait_2 \Longrightarrow \mathbf{G}\ \mathbf{F}\ crit_2)$$

This is satisfied only if some form of **fairness** is assumed.

Refinement

$\mathcal{M}'$ is **a refinement** of $\mathcal{M}$ (or it is a **realization**) of $\mathcal{M}$ if traces($\mathcal{M}'$) $\subseteq$ traces($\mathcal{M}$).

Theorem. traces($\mathcal{M}'$) $\subseteq$ traces($\mathcal{M}$) if and only if for any LT property P, $\mathcal{M} \models P$ implies $\mathcal{M}' \models P$.

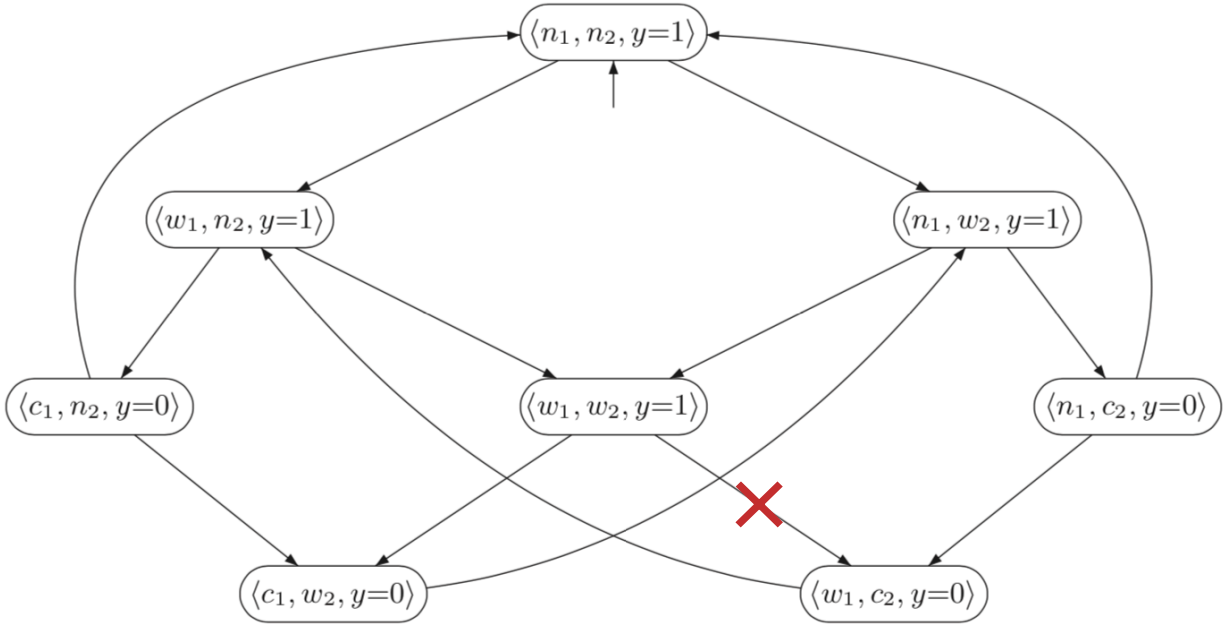
Example:
If we remove the transition from the Mutual Exclusion example:

$$\langle wait_1, wait_2, y = 1 \rangle \rightarrow \langle wait_1, crit_2, y = 0 \rangle.$$

we get a system that gives priority to P_1 (if both are waiting, P_2 cannot anymore enter its critical section).

This system has **less behaviors**.

Mutual Exclusion via semaphores



Question: does this system satisfies:

$$\mathbf{G}\,\mathbf{F}\,crit_1$$

Or

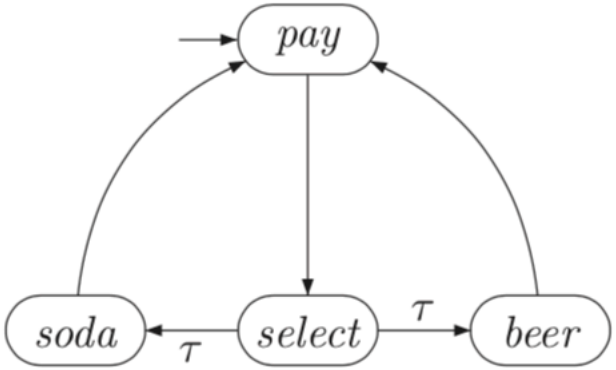
$$\mathbf{G}\,\mathbf{F}\,wait_1 \Longrightarrow \mathbf{G}\,\mathbf{F}\,crit_1$$

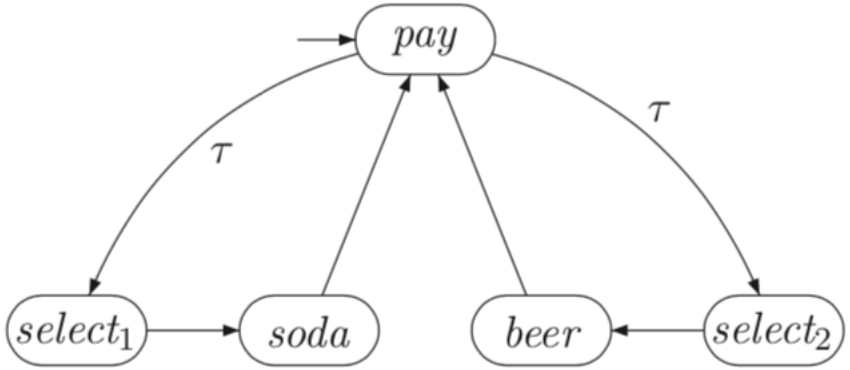
It **safisfies more** LT properties! (but not $\mathbf{G}\,\mathbf{F}\,crit_2$!!)

Equivalent Systems

Two systems $\mathcal{M}$ and $\mathcal{M}'$ are **trace-equivalent** if traces($\mathcal{M}'$) = traces($\mathcal{M}$) .

Theorem. $\mathcal{M}$ and $\mathcal{M}'$ are trace equivalent if and only they satisfy the same set of LT properties.





If $L(select) = L(select_1) = L(select_2)$, we have that **these two systems are trace equivalent**.

Invariants

An **invariant** is a safety property that depends on a condition Φ on states.

$$P_{\text{inv}} = \{ A_0 A_1 A_2 \ldots \in (2^{AP})^{\omega} \mid \text{for all } j.\ A_j \vDash \Phi \}$$

Observe that:

$$\mathcal{M} \vDash P_{inv} \text{ iff } \mathsf{traces}(\mathcal{M}) \subseteq P_{inv}$$
$$\text{iff } L(s) \vDash \Phi \text{ for all } s \text{ in a path of } \mathcal{M}$$
$$\text{iff } L(s) \vDash \Phi \text{ for all reachable states of } \mathcal{M}$$

Φ holds on **initial states** and it is **preserved** by system transitions.

Invariant Checking

Just a visit (DFS or a BFS) of the set of reachable states. During a DFS, the states on the stack is an **execution** (**counterexample**)

Algorithm 4 Invariant checking by forward depth-first search

Input: finite transition system TS and propositional formula Φ
Output: "yes" if $TS \models$ "always Φ", otherwise "no" plus a counterexample

```
set of  states R := ∅;                          (* the set of reachable states *)
stack of  states U := ε;                        (* the empty stack *)
bool b := true;                                 (* all states in R satisfy Φ *)
while (I \ R ≠ ∅ ∧ b) do
  let s ∈ I \ R;                                (* choose an arbitrary initial state not in R *)
  visit(s);                                     (* perform a DFS for each unvisited initial state *)
od
if b then
  return("yes")                                 (* TS ⊨ "always Φ" *)
else
  return("no", reverse(U))                      (* counterexample arises from the stack content *)
fi
```

If the property fails, a counterexample is provided

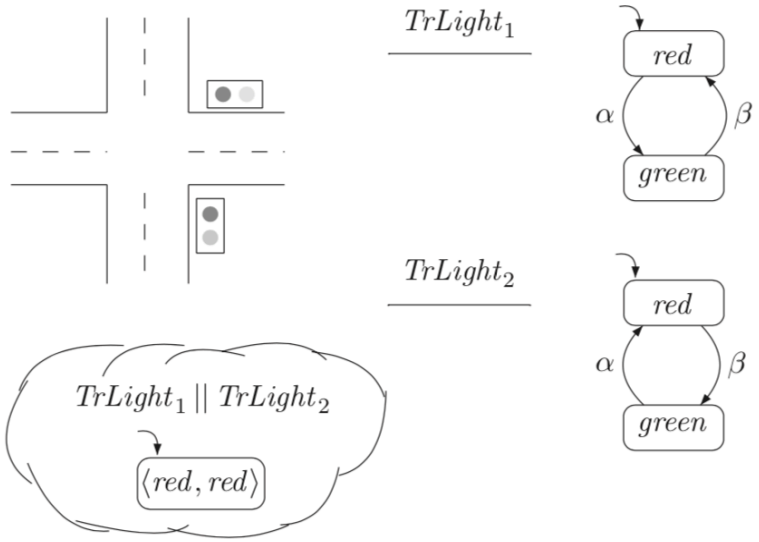
```
procedure visit (state s)
  push(s, U);                                   (* push s on the stack *)
  R := R ∪ { s };                               (* mark s as reachable *)
  repeat
    s' := top(U);
    if Post(s') ⊆ R then
      pop(U);
      b := b ∧ (s' ⊨ Φ);                        (* check validity of Φ in s' *)
    else
      let s'' ∈ Post(s') \ R
      push(s'', U);
      R := R ∪ { s'' };                         (* state s'' is a new reachable state *)
    fi
  until ((U = ε) ∨ ¬b)
endproc
```

Invariants: Deadlock

In sequential programs **termination** is a desirable property.

Often, concurrent systems are non-terminating and termination means a **deadlock**: the **system cannot evolve further.**

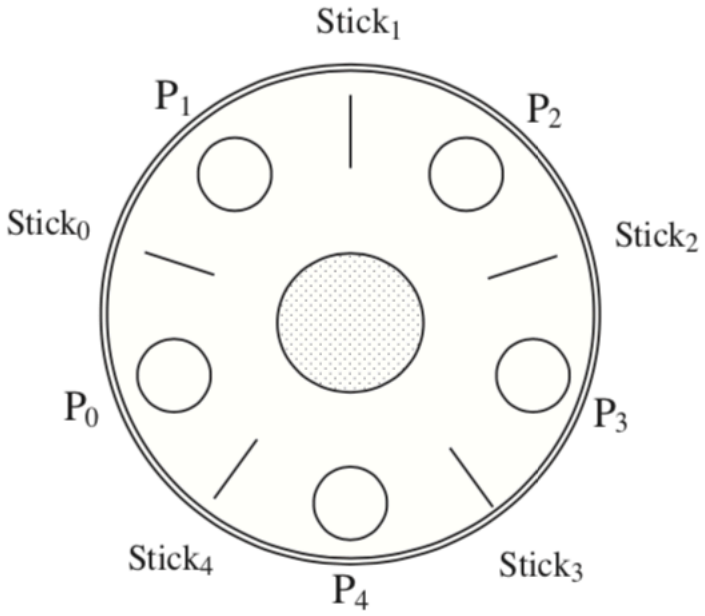
[**Observation**: in concurrent system, there exists other notions of termination, e.g, "offer an interaction after a finite time".]



Starting from the states ⟨*red*, *red*⟩ and ⟨*green*, *green*⟩, the system does not evolve (**deadlock**).

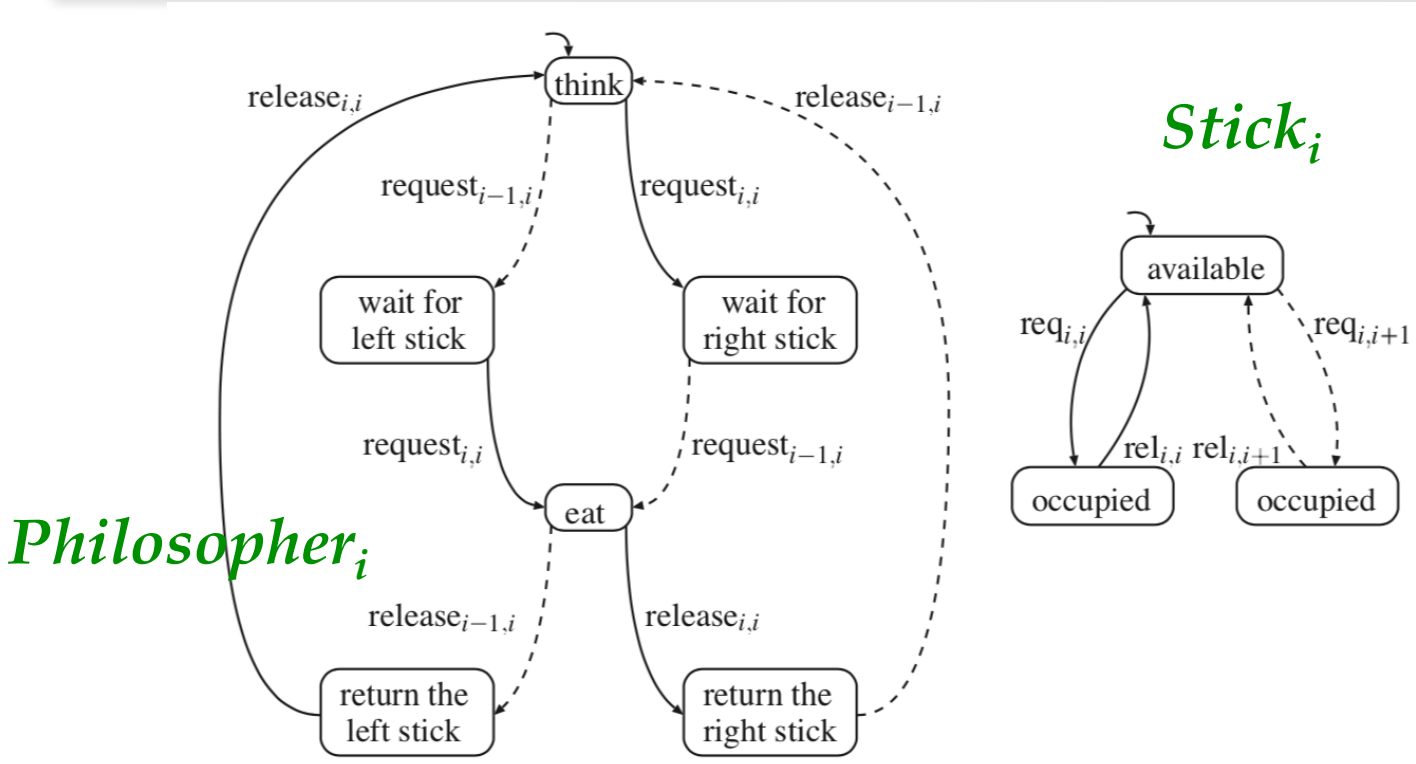
The two processes offer interactions that do not synchronize via handshaking.

Deadlock: Dining Philosophers



"Five philosophers are sitting at a round table with a bowl of rice in the middle. Their life consist in eating and thinking. To take rice, they need two chopsticks. In between two neighboring philosophers there is just one chopstick."

Deadlock prone Dining Phil.



Philosopher *i* (modelled by the TS on the left of the picture) request the chopstick on **his left** ($request_{i\text{-}1,i}$) and the one on **his right** ($request_{i,i}$) [we count modulo 5]

These actions **synchronize with corresponding actions of the process modelling the chopstick** (right in the picture)

Deadlock prone Dining Phil.

The whole system is the parallel composition:

$$Ph_4 \parallel Stick_3 \parallel Ph_3 \parallel Stick_2 \parallel Ph_2 \parallel Stick_1 \parallel Ph_1 \parallel Stick_0 \parallel Ph_0 \parallel Stick_4$$

Deadolock: All philosophers possess their left chopstick. Starting from the inital state (**all philosopher are thinking and all stick are available**):

$\langle think_4, avail_3, think_3, avail_2, think_2, avail_1, think_1, avail_0, think_0, avail_4 \rangle$

and executing the sequence of actions:

$$request_4; request_3; request_2; request_1; request_0$$

(or any permutation of them) we reach a deadlock state:

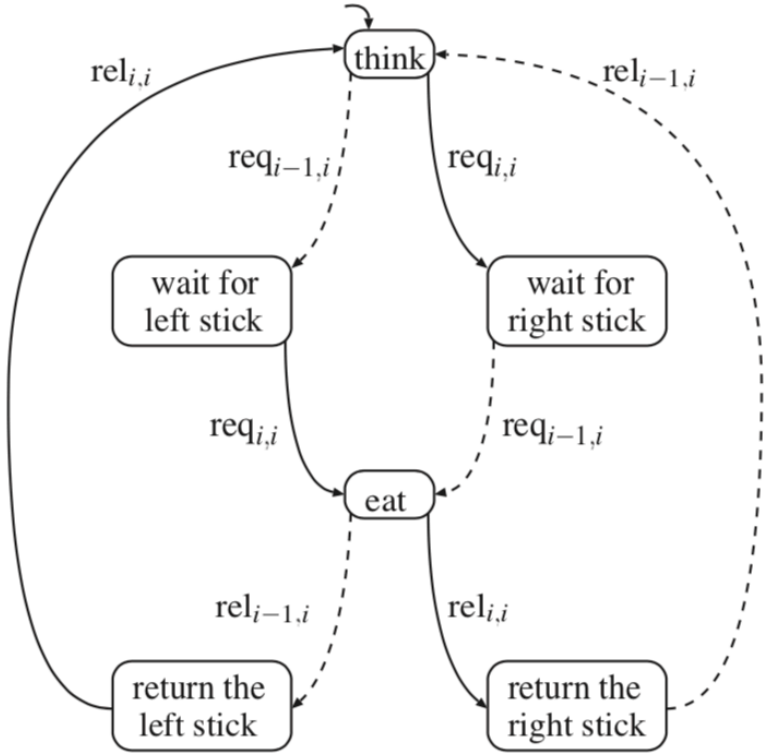
$\langle wait_{4,0}, occ_{4,4}, wait_{3,4}, occ_{3,3}, wait_{2,3}, occ_{2,2}, wait_{1,2}, occ_{1,1}, wait_{0,1}, occ_{0,0} \rangle$

Deadlock-free: At **least one** philosopher can eat and think infinitely often.

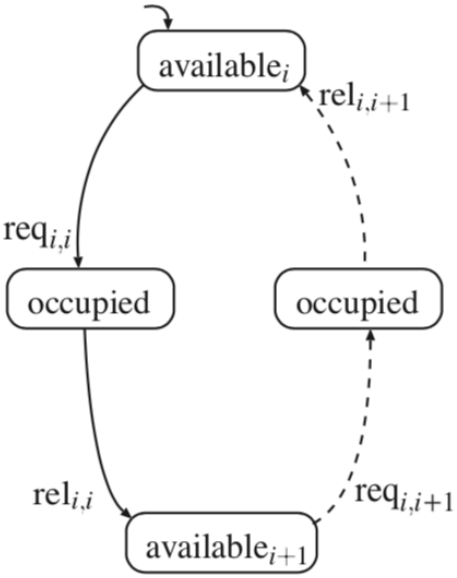
$$\mathbf{G}\,\neg\Big(\bigwedge_{0\le i<n} wait_i \wedge \bigwedge_{0\le i<n} occupied_i\Big)$$

Deadlock free Dining Philosopher

Solution: Each stick is available **for just one philosopher** at time



Two different states available.



Beyond Invariants: Safety

Definition: P is a **safety property** if for all traces σ in $(2^{AP})^{\omega} \setminus P$ there exists a set B of finite **bad prefixes** such that:

$$P \cap \{\sigma' \mid \sigma_{\text{bad}} \in B \text{ is a prefix of } \sigma'\} = \varnothing$$

Proposition. Invariants are safety properties.

Proof (sketch): Just consider finite sequences $s_0\, s_1\, s_2 \ldots\, s_n$ such that for all $i<n$ $\; s_i \vDash \Phi$ and $s_n \nvDash \Phi$. ❑

Some safety properties **are not invariants**.

Example: in the Beverage Vending Machines, let us consider the property: "The number of inserted coins is less or equal to the number of delivered drinks". The set of bad prefixes are:

{drink}, {pay, drink, drink}, …

but this property **does not depend on a specific state**.

Exercise: Could you define a transition system "equivalent" to the Beverage Vending Machine where this property is indeed an invariant?

Characterization of Safety

Lemma. If P is safety property, a system $\mathcal{M}$ satisfies P iff $\text{traces}_{\text{fin}}(\mathcal{M}) \cap B = \varnothing$, where B is the set of bad prefixes.

Definition [**Closure**].

1. $\text{pref}(\sigma) = \{\sigma' \mid \sigma' \text{ is a finite prefix of } \sigma\}$
2. $\text{pref}(P) = \bigcup_{\sigma \in P} \text{pref}(\sigma)$
3. $\text{closure}(P) = \{\sigma \mid \text{pref}(\sigma) \subseteq \text{pref}(P)\}$

Theorem. P is a safety property iff $P = \text{closure}(P)$.

Proof:

($\Rightarrow$) Let us consider the set B of bad-prefixes of P. $\text{pref}(P) \cap B = \varnothing$. This implies that all words having some prefix in $\text{pref}(P)$ belongs to P. But this is exactly $\text{closure}(P)$.

($\Leftarrow$) It is enough to show that if $P = \text{closure}(P)$ then $(2^{AP})^* \setminus \text{pref}(P)$ is the set of bad prefixes of P. ❑

Liveness

Definition: P is a **liveness property** whenever:

$$\text{pref}(P) = (2^{AP})^*$$

Intuitively: each finite word (=computation) **can be always extended** to an infinite word that satisfies P.

Proposition. The only Linear Property that is both a safety and a liveness is $(2^{AP})^{\omega}$.

Proof: If P is a liveness, $\text{pref}(P) = (2^{AP})^*$ and clearly, closure $(2^{AP})^* = (2^{AP})^{\boldsymbol{\omega}}$. If P is a safety, closure$(P)=P$. ❑

Lemma. For all linear time properties P and P':

- closure$(P \cup P')$=closure$(P) \cup$ closure(P')
- $P \subseteq$ closure(P)

Liveness & Safety

Theorem [DECOMPOSITION THEOREM]
For any linear property P, there exists a safety property P_{safe} and a liveness property P_{live} such that P = $P_{safe} \cap P_{live}$.

Proof: Any linear property P can be written as:

$$P = \text{closure}(P) \cap (P \cup ((2^{AP})^{\omega} \setminus \text{closure}(P))$$

Clearly closure(P) is a safety, and hence P_{safe} = closure(P).

We show that $P_{live} = (P \cup ((2^{AP})^{\omega} \setminus \text{closure}(P))$ is a liveness.

closure (P_{live}) = closure ($P \cup ((2^{AP})^{\omega} \setminus \text{closure}(P))$

(**Lemma**) = closure (P) $\cup$ closure($(2^{AP})^{\omega} \setminus \text{closure}(P)$)

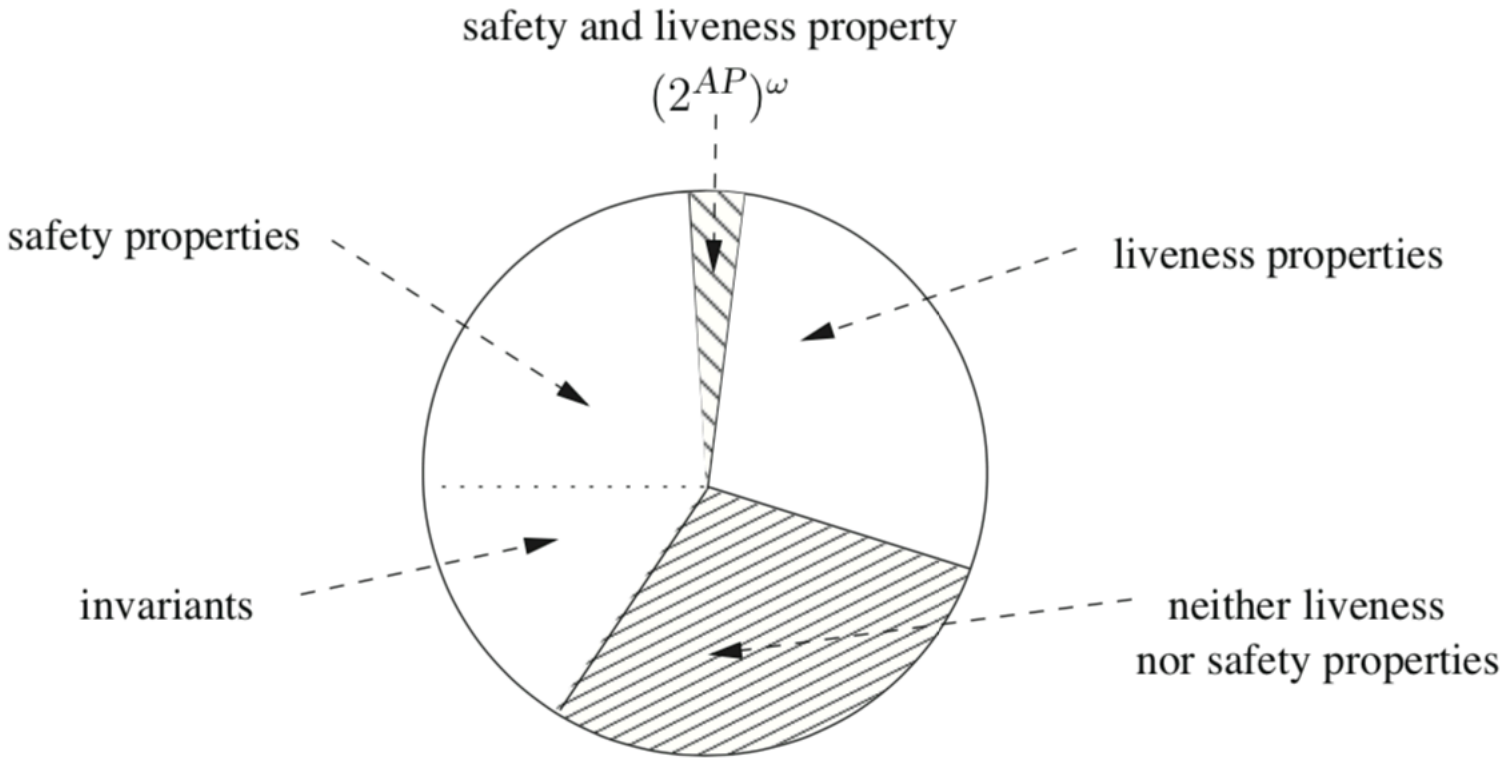
$\supseteq$ closure(P) $\cup$ $((2^{AP})^{\omega} \setminus \text{closure}(P))$

(because always P$\subseteq$closure(P))

= $(2^{AP})^{\omega}$

This implies that closure(P_{live}) = $(2^{AP})^{\omega}$ and hence pref(P_{live})= $(2^{AP})^{*}$ and therefore P_{live} is a liveness property. ❑

Liveness & Safety: summing up



Lesson 2d:

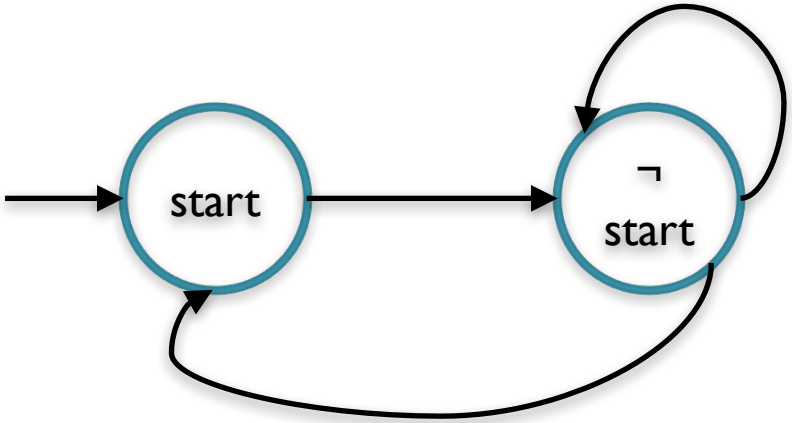
Computation Tree Logic CTL

Non Linear Time properties

"For every computation, it is always possible to return to the initial state"

A G F start

does not properly work.
It is **too strong**.



This system intuitively satisfies our intended property, but not the linear property **A G F** start (**because of the path (¬start)$^{\omega}$**)

The solution is a **branching notion** of time, allowing nesting of path quantifiers **A** and **E**: in this case **A G E F** start.

CTL: syntax

State formulas are formulas that depend on a state of a transition system

- If $p \in AP$, then p is a state formula
- If f, g are state formulas, then so are $\neg f$, $f \wedge g$, $f \vee g$
- If f is a path formula, then **A** f and **E** f are state formulas

Path formulas are formulas that depend on a computation path

- If f, g are **state** formulas, then $\neg f$, $f \wedge g$, $f \vee g$, **X** f, **F** f, **G** f, f **U** g, and f **R** g are path formulas

Similar to CTL*, but **each temporal operator** (**X**, **F**, **G**, **U**, **R**) **must be preceded by a path quantifier** (**E** or **A**)

Examples: (il)legal CTL formulas

Let $AP = \{x = 1, x < 2, x \geq 3\}$ be the set of atomic propositions.

Legal CTL formulas are:

EX $(x = 1)$, **AX** $(x = 1)$, $x = 1 \vee x < 2$

Illegal CTL formulas are:

E $(x = 1 \wedge$ **AX** $x \geq 3)$

because **AX** $x \geq 3$ is not a path formula

EX (true **U** $x = 1$)

because **EX** nested with a path formula

By contrast, the following are legal CTL formulas:

EX $(x = 1 \wedge$ **AX** $x \geq 3)$ **EX A** (true **U** $x = 1$)

Common operators: **EF** $\varphi \equiv$ "φ holds potentially"

AF $\varphi \equiv$ "φ is inevitable"

EG $\varphi \equiv$ "φ holds potentially always"

AG $\varphi \equiv$ "invariantly φ"

Minimal Fragment of CTL

From a **theoretical point of view**, only 3 operators are really needed: **EX**, **EG**, and **EU**:

$$\mathbf{AX}\, f \equiv \neg\, \mathbf{EX}\, \neg f$$

$$\mathbf{EF}\, f \equiv \neg\, \mathbf{E}\, (\text{true}\ \mathbf{U}\, f)$$

$$\mathbf{AG}\, f \equiv \neg\, \mathbf{EF}\, \neg f$$

$$\mathbf{AF}\, f \equiv \neg\, \mathbf{EG}\, \neg f$$

$$\mathbf{A}(f\, \mathbf{U}\, g) \equiv \neg\, \mathbf{E}\, (\neg g\, \mathbf{U}\, \neg f \wedge \neg g) \wedge \neg\, \mathbf{EG}\, \neg g$$

$$\mathbf{A}(f\, \mathbf{R}\, g) \equiv \neg\, \mathbf{E}\, (\neg f\, \mathbf{U}\, \neg g)$$

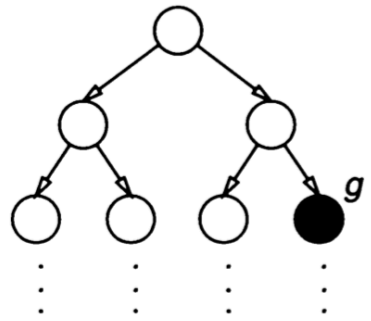
$$\mathbf{E}\, (f\, \mathbf{R}\, g) \equiv \neg\, \mathbf{A}\, (\neg f\, \mathbf{U}\, \neg g)$$

Attention! that propositional operators ($\wedge$, $\vee$, $\neg$, etc.) **cannot be applied to path formula**, so it is not true that **EG** f ≡ **E** ¬ **F** ¬*f* simply because the latter **is not** a CTL formula.

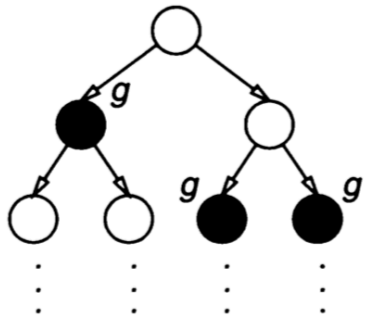
Non Linear Time (LT) examples

The semantics of CTL* formulas are relative to a computation Tree.

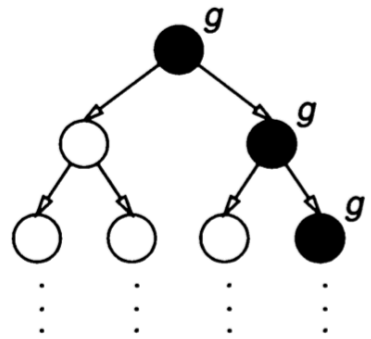
Here some example of computation trees and CTL* formulas valid in such computation trees.



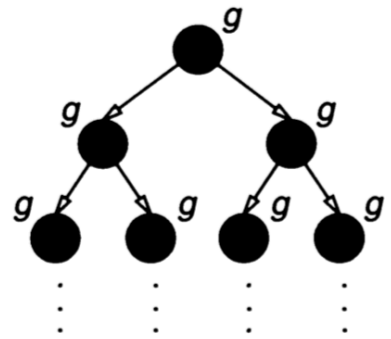
$M, s_0 \models \mathbf{EF}\, g$



$M, s_0 \models \mathbf{AF}\, g$

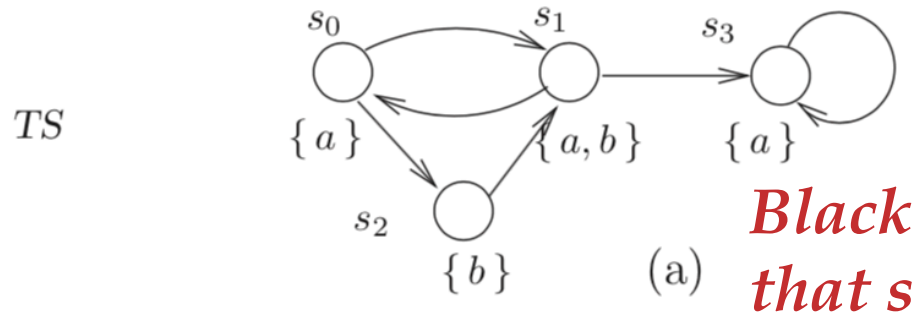


$M, s_0 \models \mathbf{EG}\, g$



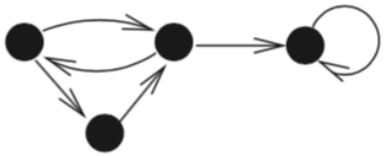
$M, s_0 \models \mathbf{AG}\, g$

Other Examples

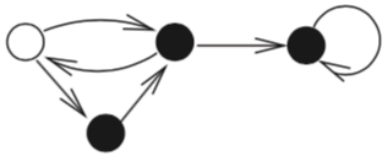


Black states are those that satisfy the formula

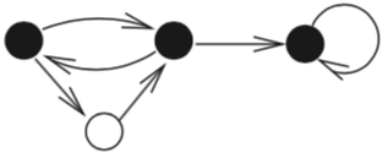
$\exists \bigcirc a$



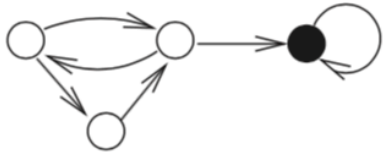
$\forall \bigcirc a$



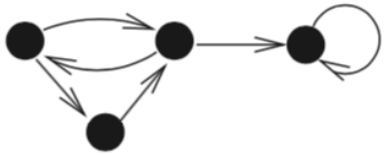
$\exists \Box a$



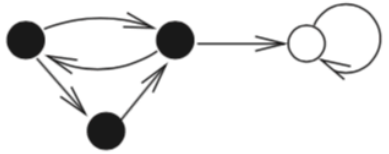
$\forall \Box a$



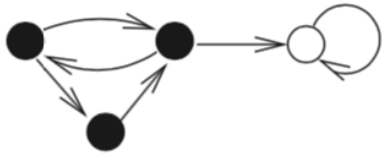
$\exists \Diamond (\exists \Box a)$



$\forall (a \mathsf{U} b)$



$\exists (a \mathsf{U} (\neg a \wedge \forall (\neg a \mathsf{U} b)))$



A remark on negation

A transition system $\mathcal{M}$ satisfies a CTL formula φ, notation $\mathcal{M} \models \varphi$ if and only if $\mathcal{M}, s \models \varphi$ for all $s \in S_0$, where S_0 is the set of initial states of $\mathcal{M}$.

Be careful that $\mathcal{M}, s \not\models \varphi$ implies $\mathcal{M}, s \models \neg\varphi$, but **it is not true** that $\mathcal{M} \not\models \varphi$ implies $\mathcal{M} \models \neg\varphi$ (**The same holds for LTL!**).

The problem is the universal quantification over initial states!

Example: Both a and $\neg a$ does not hold here:

Equivalent CTL formulas

A CTL formula f is equivalent to g if and only if for all transition system $\mathcal{M}$, $\mathcal{M} \models f$ iff $\mathcal{M} \models g$

Expansion Laws for CTL:

$$\mathbf{A}\,(f\,\mathbf{U}\,g) \equiv g \vee (f \wedge \mathbf{AX}\,\mathbf{A}(f\,\mathbf{U}\,g))$$

$$\mathbf{AG}\,f \equiv f \wedge \mathbf{AX}\,\mathbf{AG}\,f$$

$$\mathbf{AF}\,f \equiv f \vee \mathbf{AX}\,\mathbf{AF}\,f$$

$$\mathbf{E}\,(f\,\mathbf{U}\,g) \equiv g \vee (f \wedge \mathbf{EX}\,\mathbf{E}(f\,\mathbf{U}\,g))$$

$$\mathbf{EG}\,f \equiv f \wedge \mathbf{EX}\,\mathbf{EG}\,f$$

$$\mathbf{EF}\,f \equiv f \vee \mathbf{EX}\,\mathbf{EF}\,f$$

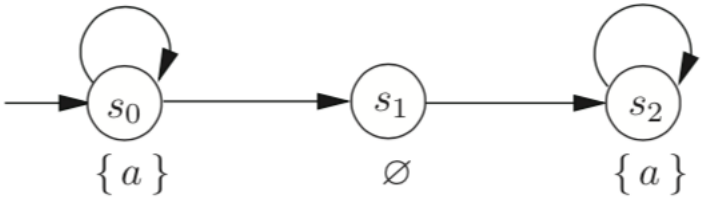
LTL versus CTL: eliminating A

Theorem. Let f be a CTL formula and let f^{LTL} be the LTL formula obtained by eliminating all path quantifiers in f. Then:

$f \equiv f^{\mathrm{LTL}}$ or there does not exist any LTL formula equivalent to f

Lemma. [PERSISTENCE] The CTL formula **A F A G** a and the LTL formula **F G** a are not equivalent.

Proof: Just consider the following Kripke structure.



We have $s_0 \models_{\mathrm{LTL}}$ **F G** a, since all path starting in s_0 will remain forever in s_0 or in s_2 (that satisfy **G** a).

By contrast $s_0 \not\models_{\mathrm{CTL}}$ **A F A G** a, since $s_0^{\boldsymbol{\omega}} \not\models_{\mathrm{CTL}}$ **F A G** a because of the paths $s_0^{*}\, s_1\, s_2^{\boldsymbol{\omega}}$ which passes the $\neg a$-state s_1. ❑

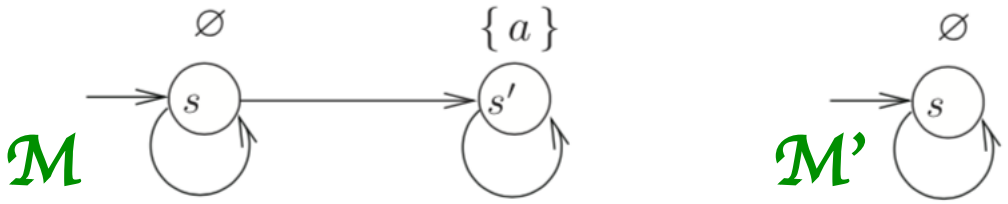
LTL and CTL are not comparable

Theorem.

1. There exist LTL formulas for which no equivalent CTL formula exist. For instance: **F G** a or **F** ($a \wedge$ **X** a)
2. There exist CTL formulas for which no equivalent LTL formula exist. For instance: **AF AG** a or **AF** ($a \wedge$ **AX** a) or **AG EF** a

Proof (idea)**:** exhibit suitable transition systems $\mathcal{M}$ and $\mathcal{M}'$ such that $\mathcal{M} \models_{\text{LTL}} g$ and $\mathcal{M}' \not\models_{\text{LTL}} g$ but such that cannot be distinguished by any CTL formula, that is, for all CTL property g, $\mathcal{M} \models_{\text{CTL}} g$ if and only if f $\mathcal{M}' \models_{\text{CTL}} g$.

Example: Let us consider **AG EF** a. This is satisfied by $\mathcal{M}$ above, but not by $\mathcal{M}'$. On the other hand since traces($\mathcal{M}'$)$\subseteq$traces($\mathcal{M}$), $\mathcal{M}'$ satisfies all LTL formulas satisfied by $\mathcal{M}$. ❑



That's all Folks!

Thanks for your attention…

…Questions?