Formal Methods in Software Development

Ivano Salvo
and Igor Melatti

Computer Science Department

SAPIENZA
UNIVERSITÀ DI ROMA

Lesson **2**, October 1st, 2019

Lesson 2A

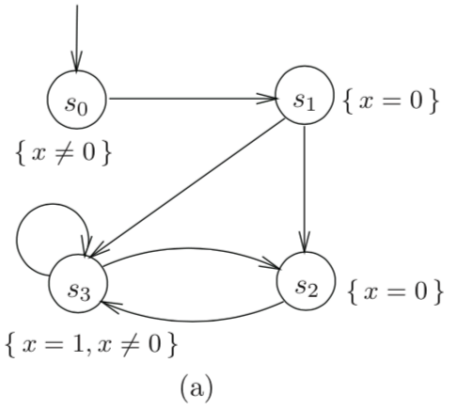
Defining Specifications

Temporal Logic

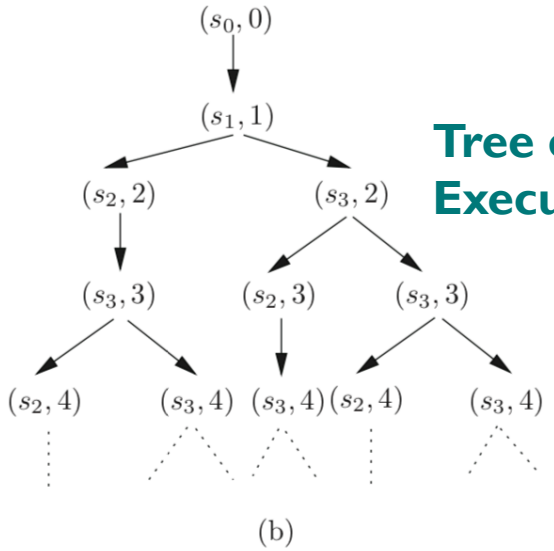
First order logic is useful to describe properties of sequential programs. Reactive/concurrent systems interact with their environment, and hence their **sequences of computation** (and its properties) are of primary importance.

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** (forall computation paths) and **E** (for some computation path). They quantify over paths starting in a given state (**state formulas**)

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

X *f* ("next time") *f* holds in the second state of a path

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

G *f* ("globally", **sometimes denoted by** $\Box$) *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.

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, the **A** f and **E** f are state formulas

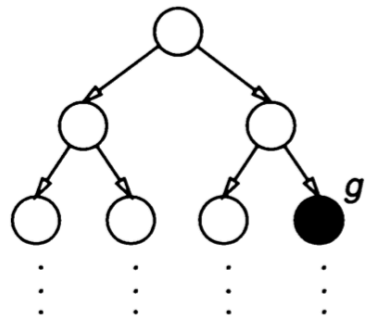
Path formulas are formulas that depend on a computation path

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

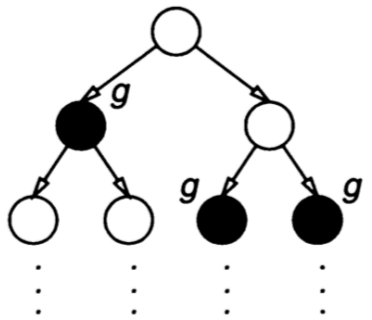
Examples

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

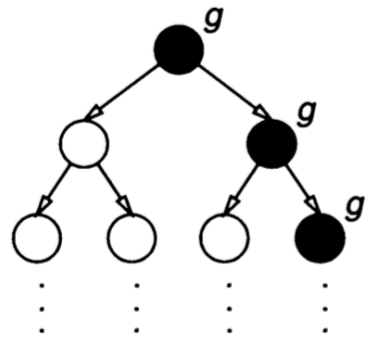
Here some example of computation trees and CTL* formulas valid in such computation trees.



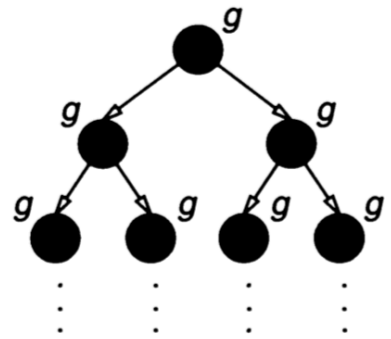
$M, s_0 \models \mathbf{EF}\, g$



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



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



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

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 $\mathcal{M}, s \models f$

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 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

$$f \wedge g \equiv \neg(\neg f \vee \neg g)$$
$$f \mathbf{R} g \equiv \neg(\neg f \mathbf{U} \neg g)$$
$$\mathbf{F} f \equiv \text{true } \mathbf{U} f$$
$$\mathbf{G} f \equiv \text{false } \mathbf{R} f$$
$$\mathbf{G} f \equiv \neg \mathbf{F} \neg f$$
$$\mathbf{A} f \equiv \neg \mathbf{E} \neg f$$

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

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

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

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

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

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

Communication: Handshaking

[Detour from lesson 1]

Another typical form of communication is via exchanging messages. Here, we see a synchronization mechanism where processes synchronize on some action. H is a set of synchronization actions.

- interleaving for $\alpha \notin H$:

$$\frac{s_1 \xrightarrow{\alpha}_1 s_1'}{\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s_1', s_2 \rangle} \qquad \frac{s_2 \xrightarrow{\alpha}_2 s_2'}{\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s_1, s_2' \rangle}$$

- handshaking for $\alpha \in H$:

$$\frac{s_1 \xrightarrow{\alpha}_1 s_1' \quad \wedge \quad s_2 \xrightarrow{\alpha}_2 s_2'}{\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s_1', s_2' \rangle}$$

processes evolve simultaneously provided they are executing the same action.

Generalising to n processes

[Detour from lesson 1]

For each pair of processes P_i, P_j there exists a set $H_{i,j}$ of actions on which they **must synchronize**.

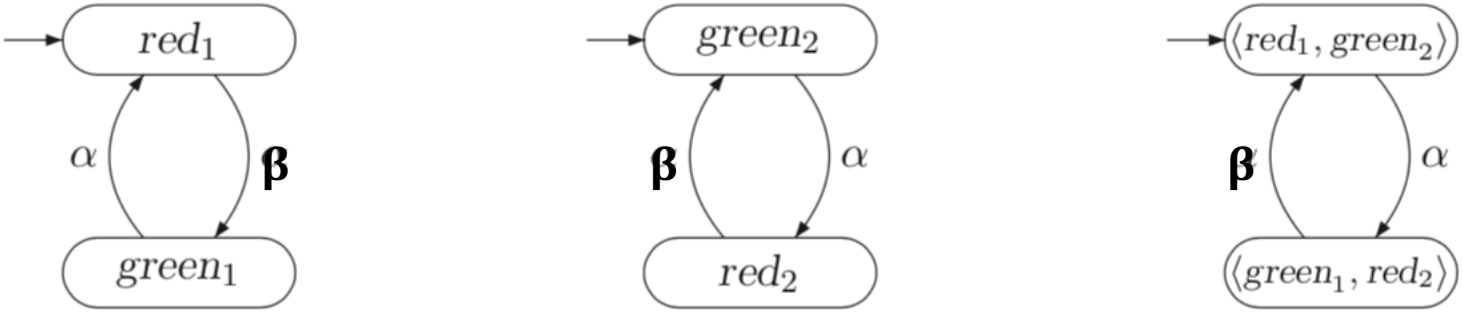
- for $\alpha \in Act_i \setminus (\bigcup_{\substack{0<j\leqslant n \\ i\neq j}} H_{i,j})$ and $0 < i \leqslant n$:

$$\frac{s_i \xrightarrow{\alpha}_i s_i'}{\langle s_1, \ldots, s_i, \ldots, s_n\rangle \xrightarrow{\alpha} \langle s_1, \ldots, s_i', \ldots s_n\rangle}$$

- for $\alpha \in H_{i,j}$ and $0 < i < j \leqslant n$:

$$\frac{s_i \xrightarrow{\alpha}_i s_i' \quad \wedge \quad s_j \xrightarrow{\alpha}_j s_j'}{\langle s_1, \ldots, s_i, \ldots, s_j, \ldots, s_n\rangle \xrightarrow{\alpha} \langle s_1, \ldots, s_i', \ldots, s_j', \ldots, s_n\rangle}$$

Example: traffic lights / 2



Two traffic lights and they parallel composition via handshaking.

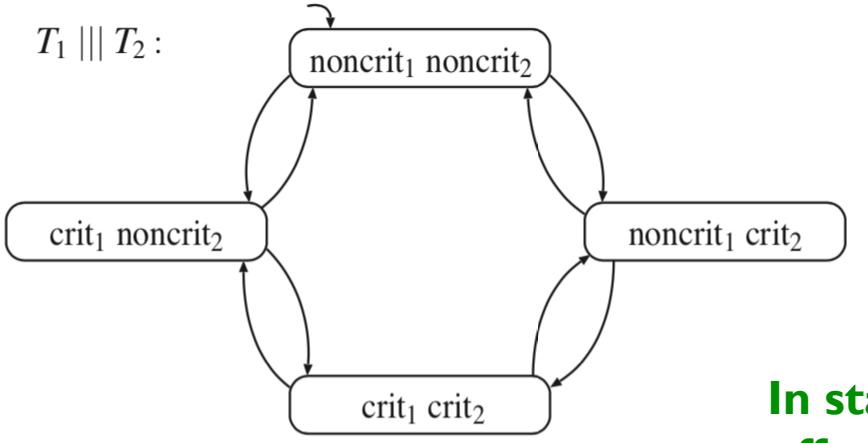
P_S = "The traffic lights are never both green simultaneously"
= (**A**) **G** ¬ ($green_1 \wedge green_2$)

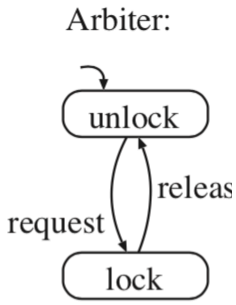
P_L = "The first traffic light will be green infinitely often"
= (**A**) **G F** $green_1$

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, $green_1$}…

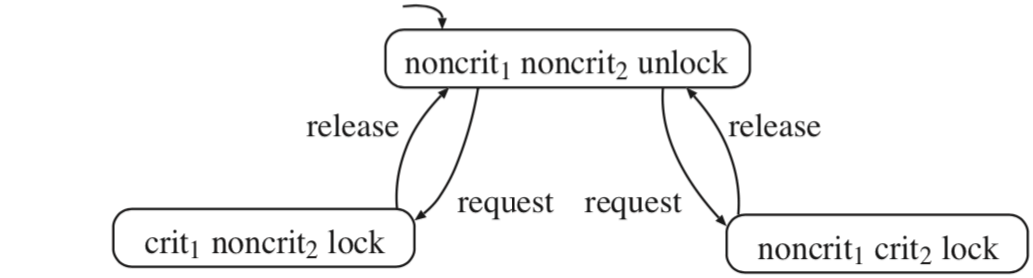
Mutual Exclusion: handshaking

Simplified version: process just have two states: *noncrit, crit*.
They synchronize with an arbiter on actions {*request*, *release*}

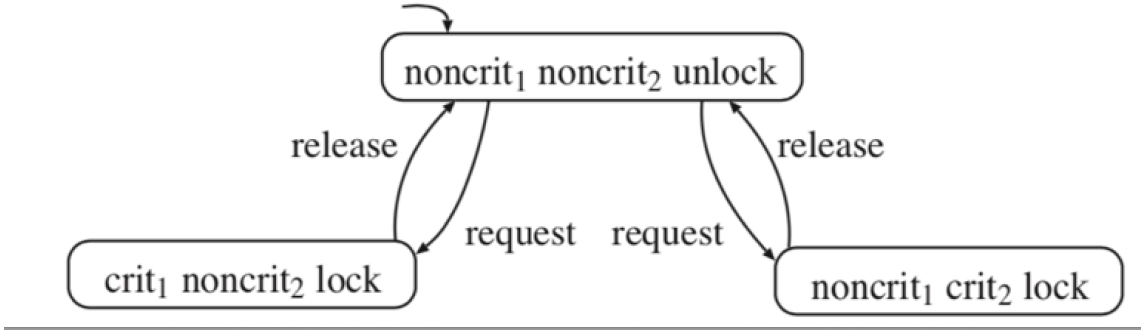




In state *lock*, the arbiter offers only the interaction *release*.



Mutual Exclusion: handshaking



This system satisfies 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:

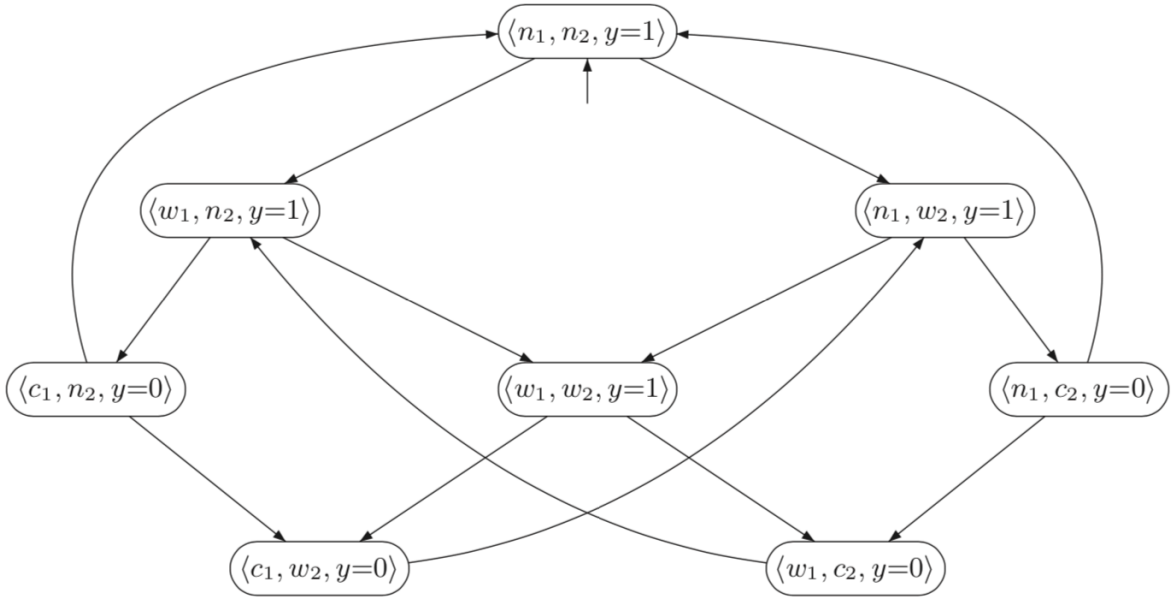
* each process enter in its critical section:

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

* each process enter 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 $\mathcal{M}' \models P$.

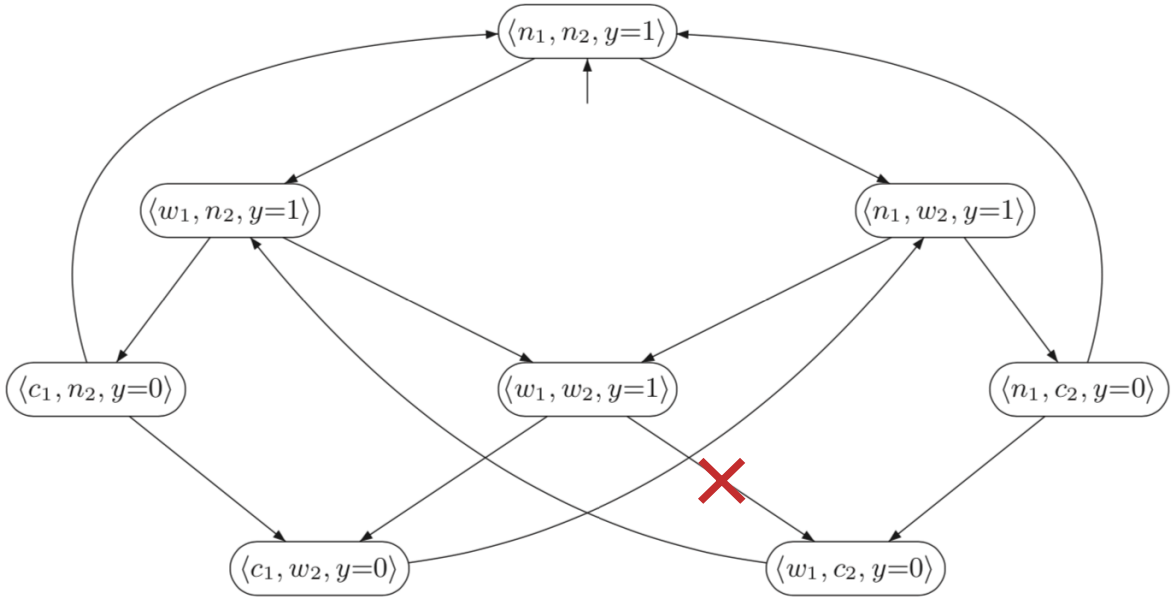
Example.
If we remove the transition from the Mutual Exclusion example:

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

We get a system that gives priority to P_2 (if both are waiting, P_1 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

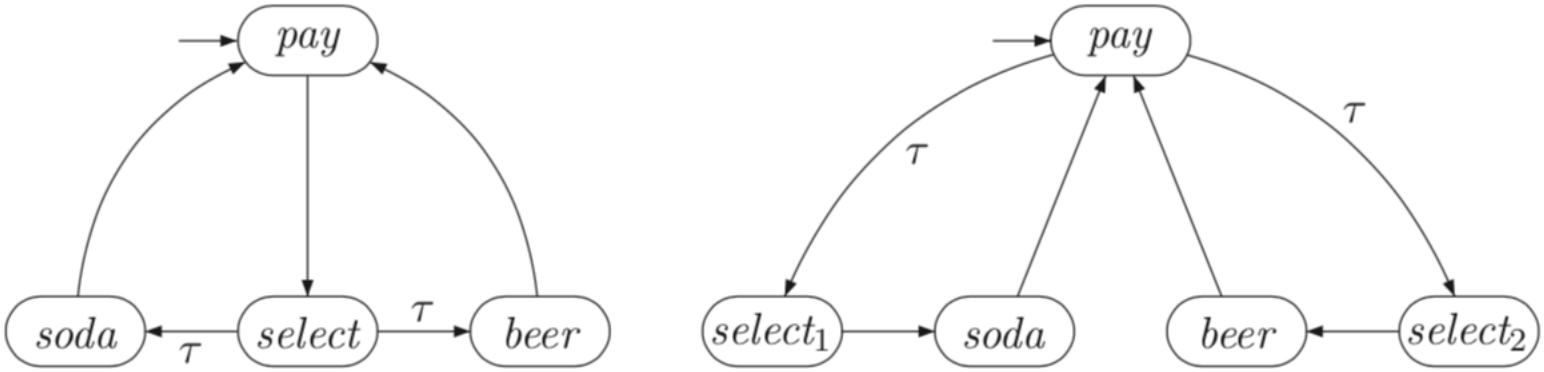
$$\mathbf{G}\,\mathbf{F}\,wait_1 \Longrightarrow \mathbf{G}\,\mathbf{F}\,crit_1$$

It **safisfies more** LT properties! (but not $\mathbf{G}\,\mathbf{F}\,crit_1$!!)

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.



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:

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

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
```

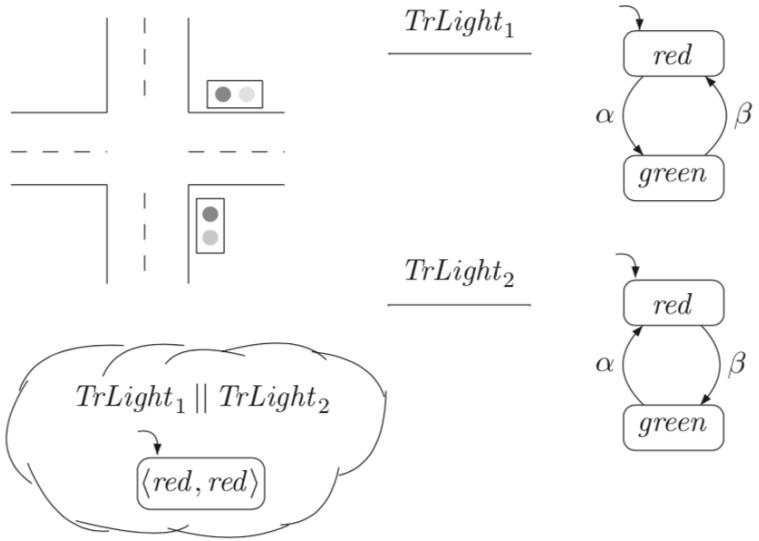
If the property fails, a counterexample is provided

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.

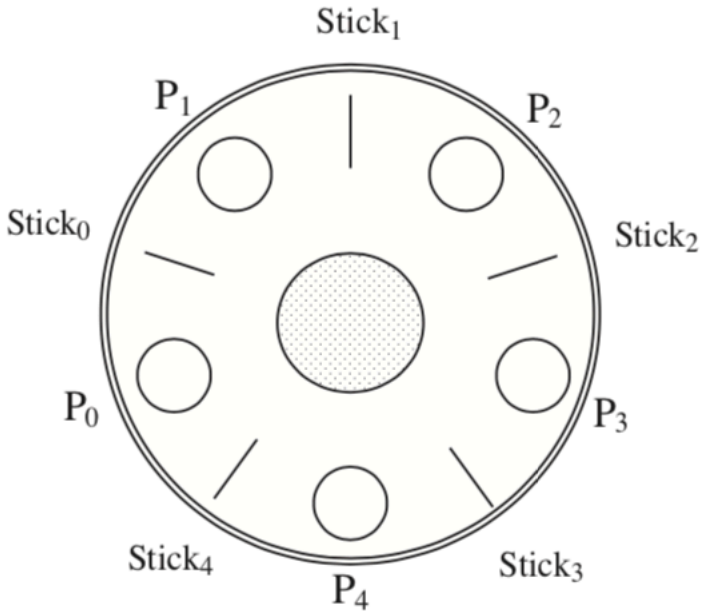
[**Observation**: in concurrent system, there exists other notions of termination, e.g, offer an interaction after a finite time.]



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

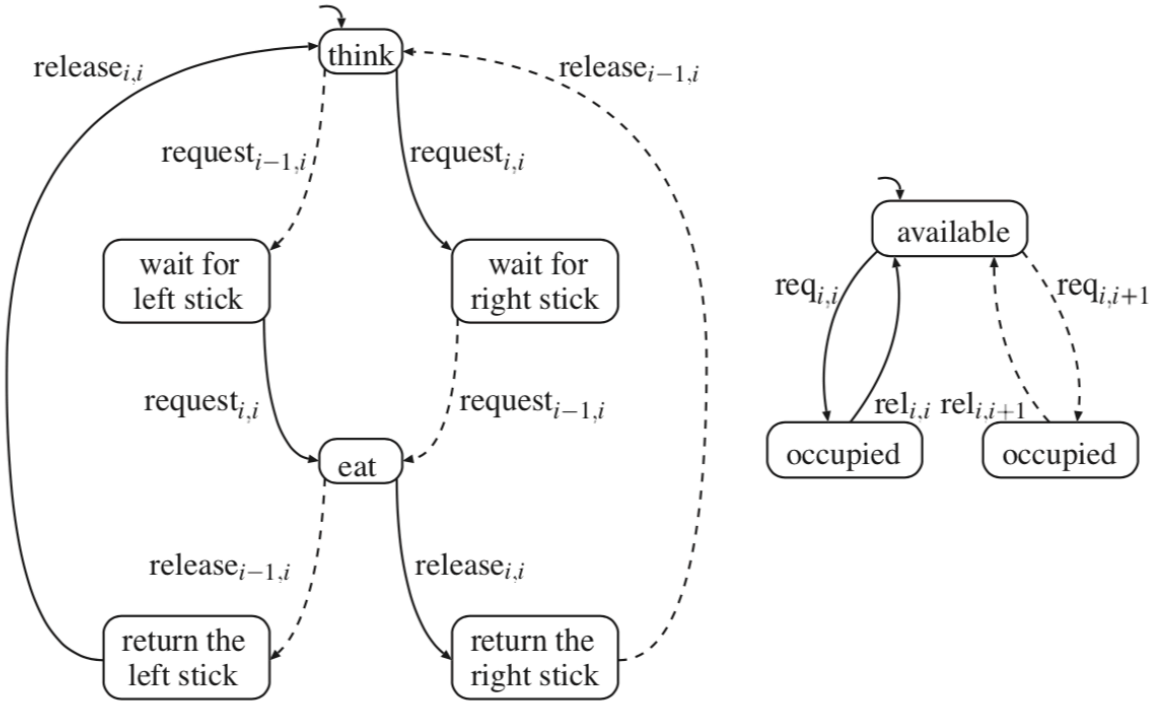
Process offer interactions that do not synchronize via handshaking.

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.

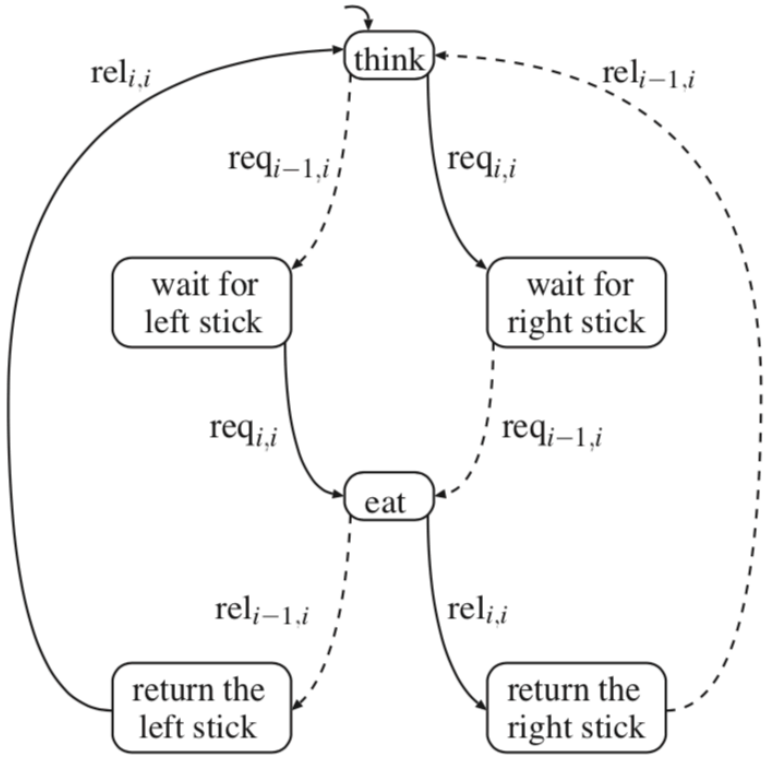


Deadolock: All philosophers possess its left chopstick.

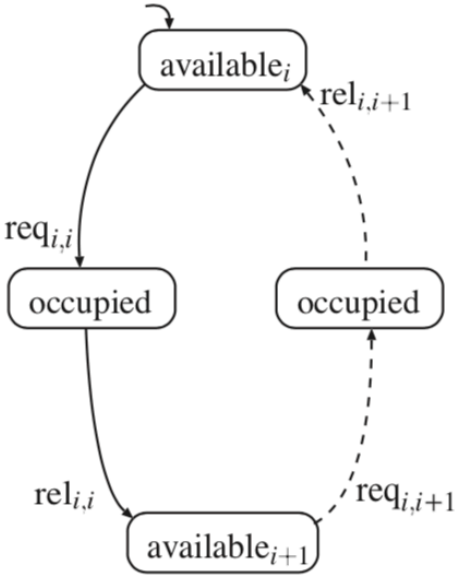
Deadlock-free: At least one philosopher can eat and think infinitely often.

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

Deadlock free Dining Phil.



Two different states available.



Solution: A chopstick is available for just one philosopher.

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 prefixes such that:

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

Proposition. Invariants are safety properties.
Just consider finite sequences $s_0\, s_1\, s_2 \ldots\, s_n$ such that for all $i<n$ $s_i \vDash \Phi$ and $s_n \nvDash \Phi$.

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

Definition.

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 $P = \text{closure}(P)$.

Liveness

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

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

Intuitively: each finite word can be 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 $P = P_{\text{safe}} \cap P_{\text{live}}$.

Proof: Any linear property P can be written as:

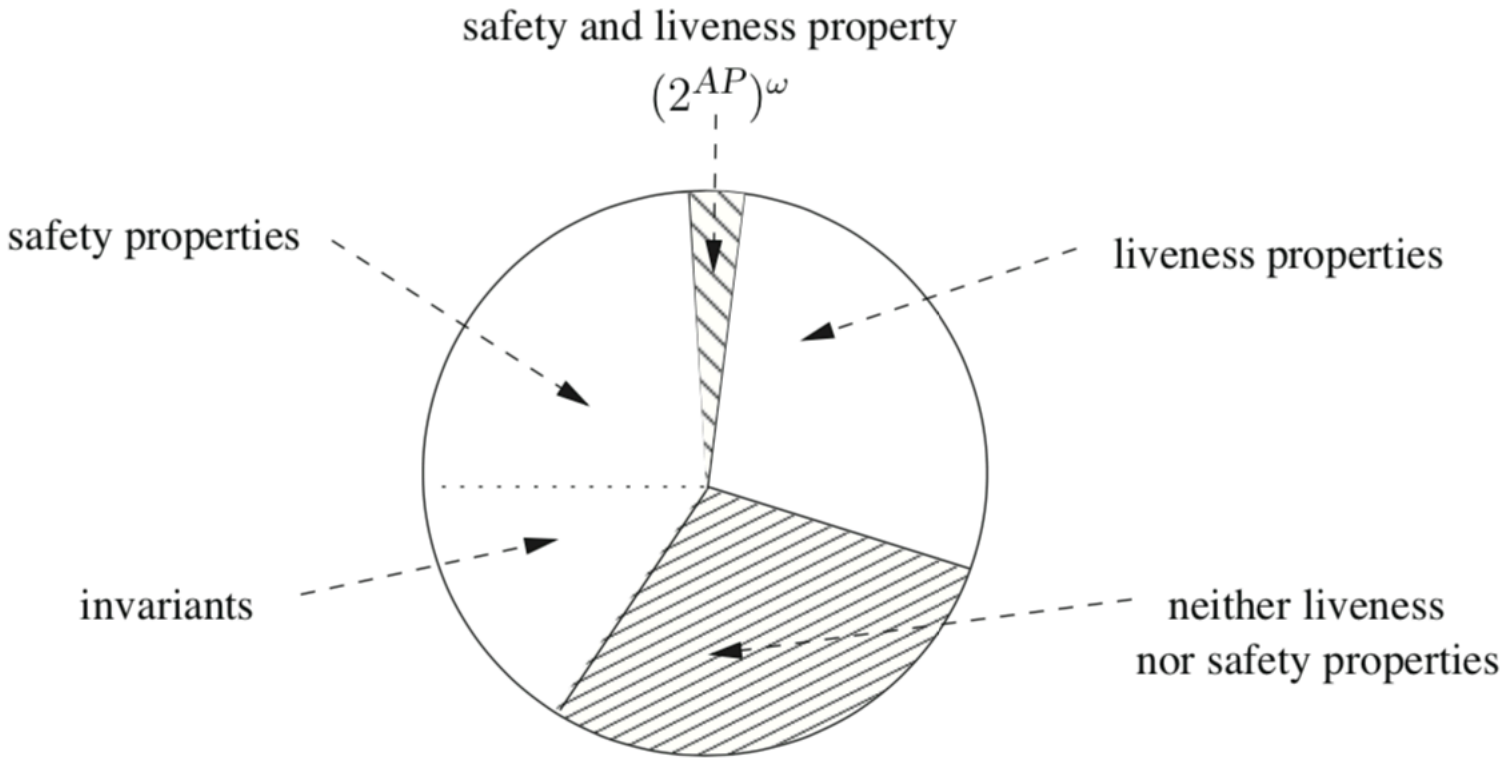
$$P = \text{closure}(P) \cap (P \cup ((2^{AP})^{\omega} \setminus \text{closure}(P))$$

Clearly closure(P) is a safety, and hence $P_{\text{safe}} = \text{closure}(P)$ and we show that $P_{\text{live}} = (P \cup ((2^{AP})^{\omega} \setminus \text{closure}(P))$ is a liveness.

$$\begin{aligned}\text{closure}(P_{\text{live}}) &= \text{closure}(P \cup ((2^{AP})^{\omega} \setminus \text{closure}(P)) \\ &= \text{closure}(P) \cup \text{closure}((2^{AP})^{\omega} \setminus \text{closure}(P)) \\ &\supseteq \text{closure}(P) \cup ((2^{AP})^{\omega} \setminus \text{closure}(P)) \\ &= (2^{AP})^{\omega}\end{aligned}$$

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

Liveness & Safety: summing up



Lesson 2B:

LTL Model Checking

Linear Time Logic (LTL)

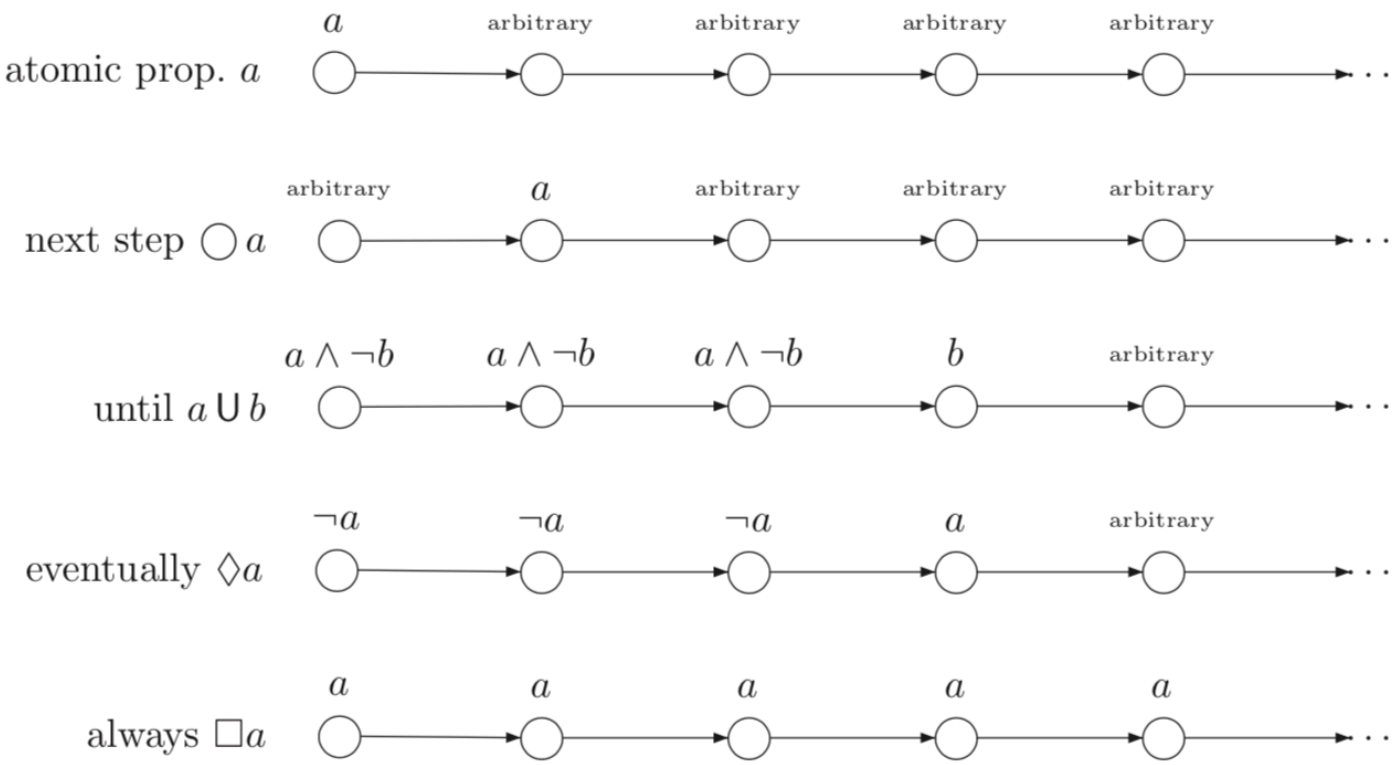
LTL is a fragment of CTL* where formulas have the form **A** *f* with *f* a path formula. Differently from CTL* path formulas are just atomic propositions (no nested occurrences of **A** o **G**)

- 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

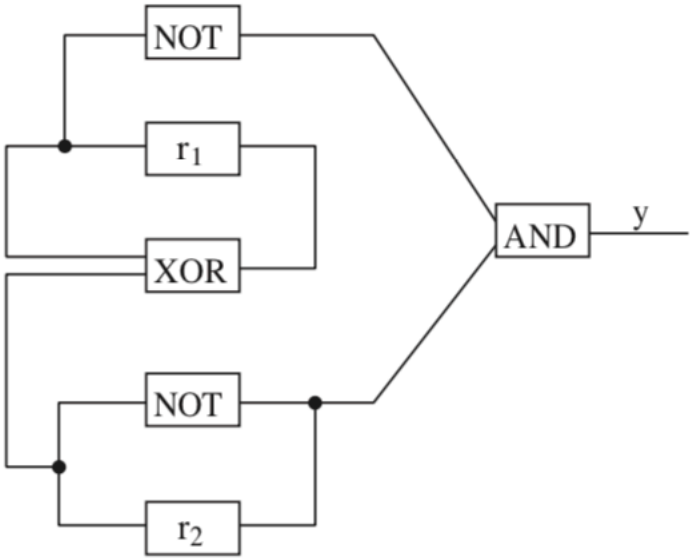
To a LTL formula φ, it is associated a LT property, defined by the set of paths π such that $\pi \models \varphi$ (see semantics of CTL* -- LTL is a sublogic of **path formulas**)

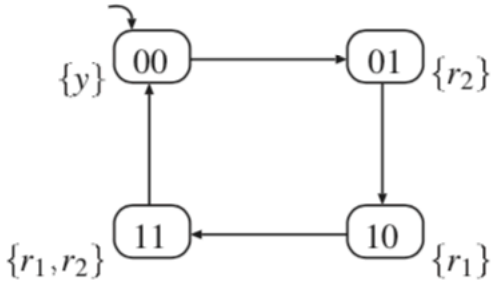
$$\mathcal{M} \models \varphi \Leftrightarrow \text{for all } s \in S_0, \mathcal{M}, s \models \varphi$$

LTL: Semantics



Ex. of neXt: modulo 4 counter





This system satisfies the property:

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

Some useful algebraic laws

Expansion Law:

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

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

Idempotency Law:

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

Absorption Law:

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

LTL Model Checking 1

Several algorithms. Today, we see a tableaux construction.

It suffices (thanks to duality) to check properties of the form $\mathbf{E}\, f$ ($\mathbf{A}\, f \equiv \neg \mathbf{E} \neg f$). Moreover (again thanks to duality), we consider only operators **X** and **U**.

Definition [Closure of f] CL(f) is the **smallest set** containing f and satisfying:

- $\neg g \in \mathrm{CL}(f)$ iff $g \in \mathrm{CL}(f)$
- If $g_1 \vee g_2 \in \mathrm{CL}(f)$ then $g_1, g_2 \in \mathrm{CL}(f)$
- If $\mathbf{X}\, g \in \mathrm{CL}(f)$ then $g \in \mathrm{CL}(f)$
- If $\neg \mathbf{X}\, g \in \mathrm{CL}(f)$ then $\mathbf{X} \neg g \in \mathrm{CL}(f)$
- If $g_1\ \mathbf{U}\ g_2 \in \mathrm{CL}(f)$ then $g_1, g_2, \mathbf{X}\,(g_1\ \mathbf{U}\ g_2) \in \mathrm{CL}(f)$

LTL Model Checking 2

Definition An **atom** is a pair (s, K), where s is a state and K is a maximal set of formulas in CL(f) consistent with $L(s)$, that is (we identify : g with $\neg\neg g$)

- for each atomic proposition p, $p \in K$ iff $p \in L(s)$
- for each $g \in$ CL(f) then $g \in K$ iff $\neg g \notin K$
- for each $g_1 \vee g_2 \in$ CL(f), $g_1 \vee g_2 \in K$ iff $g_1 \in K$ or $g_2 \in K$
- for each $\neg \mathbf{X}\, g \in$ CL(f), $\neg \mathbf{X}\, g \in K$ iff $\mathbf{X}\neg g \in K$
- for each $g_1 \mathbf{U}\, g_2 \in$ CL(f), $g_1 \mathbf{U}\, g_2 \in K$ iff $g_2 \in K$ or $g_1 \in K$ and $X(g_1 \mathbf{U}\, g_2) \in K$

Definition Given a LTL model checking problem $\mathcal{M}, s \models \mathbf{E}\, f$, the **atom graph** $G^{\mathcal{M}, f}$ is built with atoms as the set of vertices. There is an edge from (s, K) to (s', K') iff (s, s') is transition in $\mathcal{M}$, and for each formula $\mathbf{X}\, g \in$ CL(f), $\mathbf{X}\, g \in K$ iff $g \in K'$.

LTL Model Checking 3

Definition An **eventuality sequence** is an infinite path π in G such that if $g_1 \mathbf{U} g_2 \in K$ for some atom (s, K) then there exists an atom (s', K') reachable from (s, K) along π, such that $g_2 \in K'$.

Theorem $\mathcal{M}, s \models \mathbf{E} f \Leftrightarrow$ there exists an eventuality sequence starting at atom (s, K) such that $f \in K$.

Proof (sketch):
(**If**) Assume (s_0, K_0) (s_1, K_1) $(s_2, K_2)\ldots$ is an eventuality sequence with $(s, K)=(s_0, K_0)$. By def, $\pi = s_0 s_1 s_2\ldots$ is a path in $\mathcal{M}$.

To make induction hypothesis work, we prove that for every $g \in \mathrm{CL}(f)$ and for all $i \geq 0$, $\pi^i \models g$ iff $g \in K_i$. The proof proceeds by induction on sub-formulas of f.

If $g= \neg h$, $\pi^i \models g \Leftrightarrow \pi^i \not\models h \Leftrightarrow h \notin K_i$ (**IND**) $\Leftrightarrow g \in K_i$ (by maximality)

If $g=\mathbf{X} h$ then $\pi^i \models g \Leftrightarrow \pi^{i+1} \models h$ (**IND**) $h \in K_{i+1}$. Since (s_i, K_i) (s_{i+1}, K_{i+1}) $h \in K_{i+1} \Leftrightarrow \mathbf{X} h \in K_i$.

LTL Model Checking 4

Proof (cntd.):
If $g=h_1$ **U** h_2 we have $h_2 \in K_j$ for some $j \geq i$ and h_1, **X** $g \in K_k$ for $i \leq k < j$. This implies $h_1 \in K_k$ and $h_2 \in K_j$ and hence $\pi^i \models g$.

Conversely, if $\pi^i \models g$, there exists $j \geq i$ such that $\pi^j \models h_2$ and $\pi^k \models h_1$ K_k for $i \leq k < j$. (**HH**) $h_2 \in K_j$ and $h_1 \in K_k$. By absurd, $g \notin K_i$ implies that **X** $g \notin K_i$ (def of atom) and hence (def of atom) **X** $\neg g \in K_i$ (def of transition relation) $\neg g \in K_{i+1}$ and $\neg g \in K_{i+1}$ and so on until $g \notin K_j$ against the fact that $h_2 \in K_j$.

(**only if**) Assuming that $\mathcal{M}, s \models \mathbf{E}\, f$ there exists a path $\pi = s_0\, s_1\, s_2 \ldots$ in $\mathcal{M}$ such that $\pi \models f$. Define $K_i = \{\, g \in \mathrm{CL}(f) \mid \pi^i \models g \,\}$.
One can show that:

1. (s_i, K_i) is an atom;
2. $(s_i, K_i)\,(s_{i+1}, K_{i+1})$ is a transition in G.
3. The sequence $(s_0, K_0)\,(s_1, K_1)\,(s_2, K_2)\ldots$ is an eventuality sequence. ❑

LTL Model Checking 5

Definition: A non trivial strongly connected component C in G is **self-fullfilling** if for every atom (s, K) and for every $h_1 \mathbf{U} h_2 \in K$ there exists an atom (s', K') in C such that $h_2 \in K'$.

Theorem. There exists an eventuality sequence in G starting at an atom (s, K) iff there exists a self-fulfilling strongly connected component in G reachable from (s, K) .

Proof (sketch): (**If**) Take an eventuality sequence and consider the set of atoms C' that appear infinitely often in it. $C' \subseteq C$, C strongly connected component. Take (s, K) in C and $g = h_1 \mathbf{U} h_2 \in K$. There must be a path from (s, K) to C'. If h_2 appear in the path, ok. Otherwise, g is in every atom on the path and in an atom of C'. Since C' comes from an eventuality sequence, h_2 is in some atom of $C' \subseteq C$, thus C is self-fullfilling.

(**Only if**) Take a path from (s, K) to C. Clearly in C any subformula of the form $h_1 \mathbf{U} h_2$ is followed by an atom containing h_2 . The only problem is along the path, but we can reason as in the (**If**) part. ❑

--

LTL Model Checking: Algorithm

Theorem $\mathcal{M}, s \models \mathbf{E}\, f$ if and only if there exists an atom (s, K) such that $f \in K$ and a path from (s, K) to a self-fullfilling SCC.

The size of the graph G is $(|S| + |R|) \cdot 2^{|f|}$.

Using Tarjan algorithm for SCC, this is also the complexity of this algorithm for LTL model checking.

Bad News: It is exponential in the size of the formula f.

Good News: Usually the transition system is huge, but the formula is small.

Is there any polynomial algorithm for LTL model checking?

Probably, no (unless P=NP).

--

LTL Model Checking Complexity

The LTL model checking problem is PSPACE-complete.
Here we prove just that LTL model checking is *NP*-hard.

We reduce a **Hamiltonian path problem** for a graph $G=(V, E)$ to the LTL model checking problem $\mathcal{M}, s \models \mathbf{E}\, f$ where:

- $\mathcal{M}$ is the Kripke structure (S, R, L) where:
  - S is $V \cup \{s, t\}$
  - R is $E \cup \{(s, v) \mid v \in V\} \cup \{(v, t) \mid v \in V\}$
  - $L(v_i)=\{p_i\}$ and $L(s) = L(t) = \varnothing$.
- s is a state in $\mathcal{M}$, and
- f is the formula:

$$\mathbf{E}\,(\mathbf{F}\,p_1 \wedge \ldots \wedge \mathbf{F}\,p_n \wedge \mathbf{G}\,(p_1 \rightarrow \mathbf{X}\,\mathbf{G}\,\neg p_1) \wedge \ldots \wedge \mathbf{G}\,(p_n \rightarrow \mathbf{X}\,\mathbf{G}\,\neg p_n)$$

There exists a path that contains all nodes

Each node occurs just once

$\mathcal{M}, s \models \mathbf{E}\, f$ holds if and only if there exists an Hamiltonian path in G.