

# Faculty of Computer Systems & Software Engineering

## **Formal methods. Liveness and Fairness**

**Vitaliy Mezhuyev**

# Liveness property

- We have developed specifications of HourClock and Async Interface by setting restrictions on its possible states
- These are specifications of what a system *must not do* and describe so called **safety properties** of a system.  
E.g. **Len (Buf) < 3** or **hr \in (1..12)**
- Safety properties are satisfied by a **finite** behavior, which has not been violated by any step so far.
- They don't require that the system ever actually do anything.

# Liveness properties

- Lets learn how to specify *that something does happen*, i.e. that the clock keeps ticking or a message is eventually read from memory;
- We will call it *liveness properties* – the ones, that cannot be violated at any particular state;
- Only by examining an entire **infinite** behavior we can tell that the clock has stopped ticking, or that a message is never sent;
- We will express liveness properties as temporal formulas.

# Temporal Formulas

From the previous lectures we know, that:

- A state assigns a value to every variable of a system;
- A behavior is an infinite sequence of states;
- A temporal formula is true or false of a behavior.

Temporal formula ***F*** assigns a Boolean value to a behavior ***σ***

$$\sigma \models F$$

Note, in ASCII, we write ***σ*** |= ***F***

We will say that *F* is *true* of behavior *σ*, or that *σ* *satisfies* *F*, iff  $\sigma \models F$  equals true.

# Boolean combination of temporal formulas

- The formula  **$F \wedge G$**  is true of a behavior  $\sigma$  iff both  $F$  and  $G$  are true of  $\sigma$

$$\sigma \models (F \wedge G) \triangleq (\sigma \models F) \wedge (\sigma \models G)$$

- The formula  **$\neg F$**  is true of  $\sigma$  iff  $F$  is not true of  $\sigma$ .

$$\sigma \models \neg F \triangleq \neg (\sigma \models F)$$

- These are the definitions of the meaning of  $\wedge$  and of  $\neg$  as operators on temporal formulas.
- The meanings of the other Boolean operators are similarly defined.

# Temporal formulas

All the unquantified temporal formulas are Boolean combinations of three simple kinds of formulas:

- A *state predicate* (an action that contains no primed variables), is true of a behavior iff it is true *in the first state* of the behavior (e.g.  $\text{HCini} == \text{hr} \setminus \text{in } (1 \dots 12)$  ).
- A formula  $\Box P$  (in ASCII, `[]P`), where **P** is a *state predicate*, is true of a behavior iff **P** is true *in every state* of the behavior (e.g. `[]HCini` ).
- A formula  $\Box[N]_v$  (in ASCII, `[] [N]_v`) , where **N** is an *action* and *v* is a state variable, is true of a behavior iff every successive pair of steps in the behavior is a  $\Box[N]_v$  step (e.g. `[] [HCnxt]_hr` ).

# Actions as temporal formulas

Let  $\sigma_i$  – is the  $(i + 1)$  state of the behavior  $\sigma$  for any natural number  $i$ ,

so  $\sigma$  is the behavior  $\sigma_0 \rightarrow \sigma_1 \rightarrow \sigma_2 \rightarrow \dots$

Arbitrary *action*  $\mathbf{A}$  as a *temporal formula* iff  $\sigma \models \mathbf{A}$  to be true for first two states of  $\sigma$  in  $\mathbf{A}$  step.

That is, we define  $\sigma \models \mathbf{A}$  to be true iff  $\sigma_0 \rightarrow \sigma_1$  is an  $\mathbf{A}$  step.

In the special case, when  $\mathbf{A}$  is a state predicate,  $\sigma_0 \rightarrow \sigma_1$  is an  $\mathbf{A}$  step iff  $\mathbf{A}$  is true in the zero state  $\sigma_0$

# Actions as temporal formulas

$\Box[N]_v$  is true of a behavior iff each step is a  $[N]_v$  step.

So for any natural number  $n$

$\sigma \models \Box A$  to be true iff  $\sigma_n \rightarrow \sigma_{n+1}$  is an  $A$  step

In other words, for any temporal formula  $A$

$$\sigma \models \Box A \equiv \forall n \in \text{Nat} : \sigma^{+n} \models A$$

Where

$$\sigma^{+n} \triangleq \sigma_n \rightarrow \sigma_{n+1} \rightarrow \sigma_{n+2} \rightarrow \dots$$



## Temporal formulas - example

$$\sigma \models \Box((x = 1) \Rightarrow \Box(y > 0))$$

is true iff, for all  $n \in \text{Nat}$ , if  $x = 1$  is true in state  $\sigma_n$ , then  $y > 0$  is true

in all states  $\sigma_{n+m}$  with  $m \geq 0$ .

Thus, it asserts that, any time  **$x = 1$**  is true,  **$y > 0$**  is true *from then on*.

We can read  $\Box$  as ***always*** or *henceforth* or *from then on*.



# Temporal formulas - stuttering steps

Specification should allow stuttering steps, that leave unchanged all the variables appearing in the formula.

We say that a formula **F** is **invariant under stuttering** iff adding or deleting a stuttering step to a behavior  $\sigma$  does not affect whether  $\sigma$  satisfies F.

A state predicate is always invariant under stuttering, i.e. since *its truth depends only on the first state of a behavior*, and adding a stuttering step doesn't change the first state.

# Basic temporal formulas

Lets consider temporal formulas constructed from arbitrary temporal formulas F and G.

We read  $\diamond$  **eventually** (taking eventually to include now)

In ASCII,  $\langle \rangle$

$\diamond F$  is defined to equal  $\neg \Box \neg F$

$\diamond$  asserts that F is **not always false**, which means that F is *true at some time*

## Eventual action

$\Diamond\langle A \rangle_v$  is defined to equal  $\neg\Box[\neg A]_v$

where  $A$  is an action and  $v$  a state function.

We define the action  $\langle A \rangle_v$  by

$$\langle A \rangle_v \triangleq A \wedge (v' \neq v)$$

so  $\Diamond\langle A \rangle_v$  asserts that eventually an  $\langle A \rangle_v$  step occurs.

$(v' \neq v)$  is a condition that it is not stuttering step,  
i.e. value of  $v$  is changed

# Temporal formulas

$F \leadsto G$  is defined to equal  $\Box(F \Rightarrow \Diamond G)$

We read  $\leadsto$  as *leads to*.

The formula  $F \leadsto G$  asserts that whenever  $F$  is true,  $G$  is *eventually* true, that is,  $G$  is *true then or at some later time*.

## Infinitely Often (always eventually)

$\Box\Diamond F$  asserts that at all times,

$F$  is true then or at some later time.

For time 0, this implies that  $F$  is true at some time  $n_0 \geq 0$ .

For time  $n_0+1$ , it implies that  $F$  is true at some time  $n_1 \geq n_0 + 1$ .

In other words,  $\Box\Diamond F$  implies that  $F$  is *infinitely often true*.

## Eventually always

$\Diamond\Box F$  asserts that eventually (at some time),  $F$  becomes true and remains true thereafter. In other words,  $\Diamond\Box F$  asserts that  $F$  is *eventually always* true.

# Adding liveness for Hour Clock

For HourClock we can require that the clock never stops by asserting that there must be infinitely many  $HCnxt$  steps.

$$\Box \Diamond \langle HCnxt \rangle_{hr}$$

By conjoining the liveness condition to the safety specification  $HC$  we will have specification of a clock that never stops.

$$HC \wedge \Box \Diamond \langle HCnxt \rangle_{hr}$$



----- **MODULE LiveHourClock** -----

(\* Add the liveness condition to the hour clock specification of module HourClock. \*)

**EXTENDS HourClock**

(\* Conjoin the specification with the *week fairness* condition \*)

**LSpec == HC  $\wedge$  WF\_hr(HCnxt)**

(\* Define some properties that LSpec satisfies. \*)

(\* Asserts that infinitely many  $\langle\langle\text{HCnxt}\rangle\rangle_{\text{hr}}$  steps occur. \*)

**AlwaysTick ==  $[\langle\langle\text{HCnxt}\rangle\rangle_{\text{hr}}$**

(\* Asserts that, for each time  $n$  in  $1..12$ ,  $\text{hr}$  infinitely often equals  $n$ . \*)

**AllTimes ==  $\forall n \in 1..12 : [\langle\text{hr} = n\rangle]$**

**TypeInvariance ==  $[\text{HCini}]$**

(\* The temporal formula asserting that HCini is always true. \*)

(\* It is stated in this way to show another way of telling TLC to check an invariant. \*)






-----  
**THEOREM LSpec  $\Rightarrow$  AlwaysTick  $\wedge$  AllTimes  $\wedge$  TypeInvariance**

=====

# TLC - checking temporal properties

Model Overview | Advanced Options | Model Checking Results

**Model Overview**

What is the behavior spec?

☐ Initial predicate and next-state relation

Init:

Next:

☒ Temporal formula

☐ No Behavior Spec

What to check?

☒ Deadlock

+

Invariants

-

Properties

Temporal formulas true for every possible behavior.

☒ AlwaysTick

☒ AllTimes

☒ TypeInvariance

Add

Edit

Remove

# Temporal properties of LiveHourClock

When **hr** is equal to 1, it implies that **hr** eventually will have value 2

**New** ==  $(hr = 1) \Rightarrow \langle \rangle (hr = 2)$                       \\* True of False?

When **hr** is equal to 1, it implies that **hr** always will have value 2.

**New** ==  $(hr = 1) \Rightarrow [](hr = 2)$                       \\* True of False?

If this property *infinitely often* true

**New** ==  $(hr = 1) \Rightarrow []\langle \rangle (hr = 2)$                       \\* True of False?

If this property *eventually always* true

**New** ==  $(hr = 1) \Rightarrow \langle \rangle [](hr = 2)$                       \\* True of False?

THEOREM  $LSpec \Rightarrow AlwaysTick \wedge AllTimes \wedge TypeInvariance \wedge \mathbf{New}$

## ENABLED action

In any behavior satisfying the safety specification HC, it's always possible to take an HCnxt step *that changes hr*.

Action  $\langle HCnxt \rangle_{hr}$  is therefore always enabled, so  $\text{ENABLED } \langle HCnxt \rangle_{hr}$  is true throughout such a behavior.

Formally:

$$\Box(\text{ENABLED } \langle HCnxt \rangle_{hr} \Rightarrow \Diamond \langle HCnxt \rangle_{hr})$$

In general, **ENABLED A** is a predicate that is true iff action A is enabled, meaning *that it is possible to take A step*, that changes state variables.

# General liveness condition - Weak Fairness

The general liveness condition for an action A

$$\Box(\text{ENABLED } \langle A \rangle_v \Rightarrow \Diamond \langle A \rangle_v)$$

This condition asserts that, if A ever becomes enabled, then an A step will *eventually occur*

Next formula is called **Weak Fairness**  $\text{WF}_v(A)$

$$\Box(\Box \text{ENABLED } \langle A \rangle_v \Rightarrow \Diamond \langle A \rangle_v)$$

- This formula asserts that, if A *ever becomes forever enabled, then an A step must eventually occur.*
- **WF** stands for Weak Fairness, and the condition  $\text{WF}_v(A)$  is called weak fairness on A.

# Strong Fairness

$$SF_v(A)$$

If  $A$  is infinitely often enabled, then infinitely many  $A$  steps occur.

$$\Box \Diamond \text{ENABLED } \langle A \rangle_v \Rightarrow \Box \Diamond \langle A \rangle_v$$

# Temporal Tautologies

The temporal tautology  $\Box F \Rightarrow F$  asserts the obvious fact that, if  $F$  is true at all times, then it's true at time 0.

$$\neg \Box F \equiv \Diamond \neg F$$

$F$  is not always true iff it is eventually false.

$$\Box (F \wedge G) \equiv (\Box F) \wedge (\Box G)$$

$F$  and  $G$  are both always true iff  $F$  is always true and  $G$  is always true.

Another way of saying this is that  $\Box$  distributes over  $\wedge$ .

$$\Diamond (F \vee G) \equiv (\Diamond F) \vee (\Diamond G)$$

$F$  or  $G$  is eventually true iff  $F$  is eventually true or  $G$  is eventually true.

Another way of saying this is that  $\Diamond$  distributes over  $\vee$ .

# Temporal Tautologies

The operator  $\Box$  doesn't distribute over  $\vee$ , nor does  $\Diamond$  distribute over  $\wedge$ . For example,  $\Box((n \geq 0) \vee (n < 0))$  is not equivalent to  $(\Box(n \geq 0) \vee \Box(n < 0))$ ; the first formula is true for any behavior in which  $n$  is always a number, but the second is false for a behavior in which  $n$  assumes both positive and negative values.

$\Box\Diamond$  distributes over  $\vee$  and  $\Diamond\Box$  distributes over  $\wedge$ :

$$\Box\Diamond(F \vee G) \equiv (\Box\Diamond F) \vee (\Box\Diamond G) \qquad \Diamond\Box(F \wedge G) \equiv (\Diamond\Box F) \wedge (\Diamond\Box G)$$

The first asserts that  $F$  or  $G$  is true infinitely often iff  $F$  is true infinitely often or  $G$  is true infinitely often.






# Model checking example:

## Mutual exclusion

- The mutual exclusion problem (mutex)
  - Avoiding the simultaneous access to some kind of resources by use of the *critical sections* of concurrent processes
- The problem is to find a *protocol* for determining which process is allowed to enter its critical section
- Some expected properties for a correct protocol: Safety, Liveness, Non-blocking, No strict sequencing

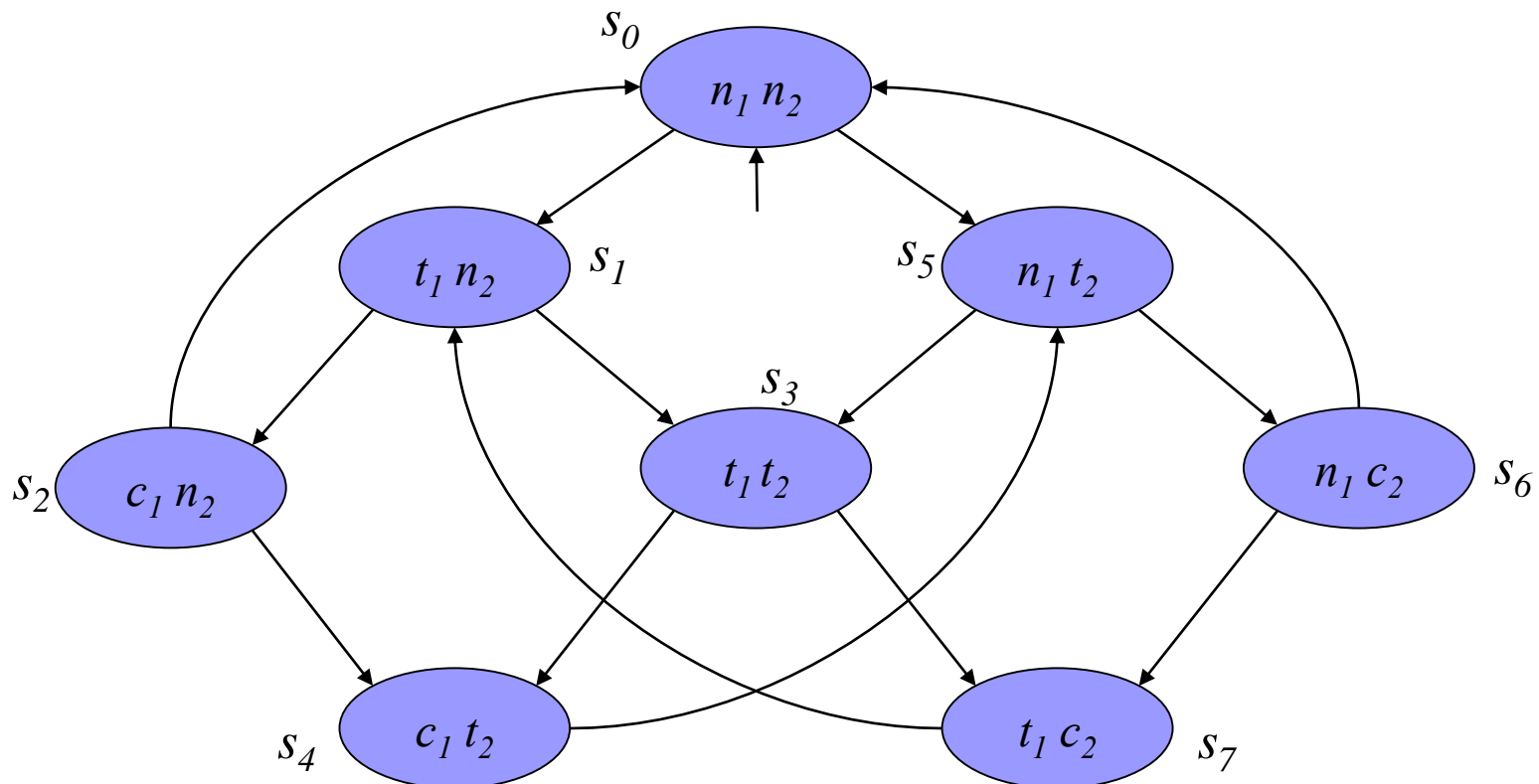
- 
- Safety: Only one process is in its critical section at any time.
  - Liveness: Whenever any process requests to enter its critical section, it will eventually be permitted to do so.
  - Non-blocking: A process can always request to enter its critical section.
  - No strict sequencing: Processes not need enter their critical section in a strict sequence.

# Modeling mutex

- Consider each process to be either in its non-critical state  $n$ , trying to enter the critical section  $t$  or  $c$
- Each individual process has this cycle:
  - $n \rightarrow t \rightarrow c \rightarrow n \rightarrow t \rightarrow c \rightarrow n \dots$
- The processes phases are interleaved
- We will define  $n$ ,  $t$ ,  $c$  as temporal formulas

# 2 process mutex

- The processes are *asynchronous interleaved*
  - one of the processes makes a transition while the other remains in its current state



- Safety: Only one process is in its critical section at any time:

$$[] \neg (c_1 \wedge c_2)$$

- Liveness: Whenever any process requests to enter its critical section, it will eventually be permitted to do so:


$$[] (n_1 \Rightarrow \diamond t_1)$$

- Non-blocking: A process can always request to enter its critical section:

$$\square [] (\exists t_1 : n_1 \Rightarrow t_1)$$

- No strict sequencing: Processes not need enter their critical section in strict sequence:

$$\square [] \diamond (c_1 \leadsto t_1) \wedge [] \diamond (c_2 \leadsto t_2)$$



**Thank you for your attention!**  
**Please ask questions**