Faculty of Computer Systems & Software Engineering

Formal methods.

Introduction to the Temporal Logic of Actions

Vitaliy Mezhuyev

History and motivation

- Why formal methods are needed?
- What is mathematical base of formal methods?
- Specification of modern computer systems needs expression of its temporal properties.
- In 1977, Amir Pnueli introduced the use of temporal logic for describing system behaviors.
- In the late 1980's, Lesly Lamport invented the Temporal Logic of Actions (TLA), a variant of Pnueli's original logic.

History and motivation

- TLA is applicable for specifying a wide class of software systems from simple programs to large *distributed* and *concurrent* systems.
- TLA is good for describing *asynchronous* systems (systems with components that do not operate in strict lock-step manner).
- TLA allows to write a precise and formal description of almost any kind of *discrete* system by a single formula.
- TLA uses first order logic and set theory for expressing ordinary mathematics.
- TLA expands ordinary mathematics by temporal operators (like *next*, *always*, *eventually*, etc.).

What we will learn with TLA

- how to specify the **safety** properties of systems (what the system should not do, practically with no temporal logic).
- how to specify **liveness** properties of systems (what the system should do - with temporal logic).
- how to specify **real-time** properties of systems with temporal logic.
- TLA+ tools: mostly, **TLC** model checker.
- The basic book: “Specifying Systems” of Leslie Lamport

References to TLA framework

The TLA Home Page:
http://research.microsoft.com/en-us/um/people/lamport/tla/tla.html

TLA+ Tools: http://research.microsoft.com/en-us/um/people/lamport/tla/tools.html

Book of Lamport: http://research.microsoft.com/en-us/um/people/lamport/tla/book.html

The Specification in TLA+

Can be written in two formats:

- the ASCII
- the TLATEX (mathematical) notation

TLATEX	ASCII	Name
∧	/\	And
∨	\/	Or
¬	~	Not
⇒	=>	Imply
≡	<=>	Equivalence
≜	==	Is defined to equal
□	[]	Box
∈	\in	In
≠	#	Not equal

Introduction to TLA

- A system specification in TLA consists of ordinary mathematics (sets, FOL) linked together with temporal logic.
- To write a speciation, we need to learn have to express ordinary math with TLA.
- So lets repeat basic statements of elementary algebra, propositional logic and set theory.

Basic math

- Elementary algebra is the mathematics of real numbers and the operations + (addition), - (subtraction), * (multiplication), and / (division).
- Propositional logic is the mathematics of the two Boolean values **true** and **false** and the operations whose names are

$\wedge$	conjunction (and)	$\Rightarrow$	implication (implies)
$\vee$	disjunction (or)	$\equiv$	equivalence (is equivalent to)
$\neg$	negation (not)		

Definition of the Boolean operators

$\wedge$ $F \wedge G$ equals TRUE iff both F and G equal TRUE.

$\vee$ $F \vee G$ equals TRUE iff F or G equals TRUE (or both do).

$\neg$ $\neg F$ equals TRUE iff F equals FALSE.

$\Rightarrow$ $F \Rightarrow G$ equals TRUE iff F equals FALSE or G equals TRUE (or both).

$\equiv$ $F \equiv G$ equals TRUE iff F and G both equal TRUE or both equal FALSE.

Iff means - if, and only if

Truth tables

Definition of the Boolean operators can be done by truth tables. For example, for the implication

F	G	$F \Rightarrow G$
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	TRUE
FALSE	FALSE	TRUE

The sense of implication

Implication **F => G** means F implies G or, equivalently, if F then G.

Example:

if n is greater than 3, then it should be greater than 1, so n > 3 should imply n > 1. Therefore, the formula

(n > 3) => (n > 1) is true.

For n = 4

(4 > 3) => (4 > 1) is true.

For n = 2

(2 > 3) => (2 > 1) is true.

For n = 0

(0 > 3) => (0 > 1) is true.

Propositional-logic formulas

Just like formulas of algebra, formulas of propositional logic are made up of values, operators, and variables (like **x)**, which containing values.

Propositional-logic formulas use only the two values:
true and **false**
and the five Boolean operators:
/, **\\/**, **¬**, **=>**, and **≡**.

Precedence

In algebraic formulas, ***** has higher precedence (binds more tightly) than +, so **x + y*z** means **x + (y*z)**.

Similarly, ¬ has higher precedence than /\ and \/, which have higher precedence than => and ≡, so **¬F/\G=>H** means **((¬F)/\G))=>H**.

Other mathematical operators like + and > have higher precedence than the operators of propositional logic, so **n>0 => n-1>=0** means **(n>0) => (n-1>=0)**.

The rule is: always use parenthesis, if you don't remember precedence.

Sets.

- Set is a collection of elements. $x \in S$ means that x is an element of S
- A set can have a finite or infinite number of elements.
- A set is completely determined by its elements. Two sets are equal if they have the same elements.
- The empty set, which has no elements, we will designate { }.
- The common operations on sets are

$\cap$ intersection $\quad$ $\cup$ union

$\subseteq$ subset $\quad$ $\setminus$ set difference

Predicate Logic

Predicate logic extends propositional logic with the two quartiers

$$
\begin{array}{ll}
\forall & \text{universal quantification (for all)} \\
\exists & \text{existential quantification (there exists)}
\end{array}
$$

These allow to say that a formula is true for all the elements of a set, or for some of the elements of a set

TLATEX	ASCII	Name
∀	\A	For all
∃	\E	There exists

Formulas of Predicate Logic

The formula $\forall x \in S : F$ asserts that formula F is true for every element x in the set S.

The formula $\exists x \in S : F$ asserts that formula F is true for at least one element x in S.

Formula F is true for some x in S iff F is not false for all x, in S, i.e., if it's not the case that ¬F is true for all x in S.

$$(\exists x \in S : F) \quad \equiv \quad \neg(\forall x \in S : \neg F)$$

Such the formulas are called *tautologies*, meaning that it is true for all values of the identifiers (sets) S and F

Specifying behavior of a system

- To describe behavior of a system we use equations that determine how its state evolves with time, where the state consists of the values of variables.

- For example, the behavior of the earth-moon system can be described by a function F from time to states, where F(t) represents the state of the system at time t.

- State of a computer system changes in discrete steps. So, we represent the **behavior of a system as a sequence of states, where a state is defined by assignment of values to variables.**

An Hour Clock System

- Let's specify a simple system - a digital clock that displays only the hour.
- The value of the hour changes through the values 1 through 12.
- Let the variable **hr** represent the clock's display. We can present a ***behavior*** of the clock as the sequence

$$[hr = 11] \rightarrow [hr = 12] \rightarrow [hr = 1] \rightarrow [hr = 2] \rightarrow \cdots$$

- where e.g. **[hr = 11]** is a state in which the variable **hr** has the value **11**.
- A pair of successive states, such as [hr = 1] -> [hr = 2], we will call a **step**.

An Hour Clock – initial predicate

- To specify the hour clock, we describe all its possible behaviors.
- We write an initial predicate that species the initial values of **hr**, and a next-state relation that species how the value of **hr** can change in any step.
- Initially, **hr** can have any value from 1 to 12. Lets fix it in initial predicate **HCini**

$$HCini \triangleq hr \in \{1, \ldots, 12\}$$

- The symbol $\triangleq$ means is defined HCini to equal.

Note, in ASCII the symbol $\triangleq$ is represented by ==

An Hour Clock – next state predicate

- The next-state predicate **HCnxt** is a formula expressing the relation between the values of **hr** in the previous (old) state and the next (new) state of a system.

- Let **hr** represent the value of **hr** in the old state and **hr**’ represent its value in the new state.

- The symbol **’** in **hr’** is read ***prime***.

- The next-state relation is that **hr ’** equals **hr+1** except if **hr** equals **12**, in which case **hr ’** should equal **1**.

An Hour Clock – next state predicate

- Using typical If/Then/Else constructs, we can define **HCnxt** to be the next-state relation

$$HCnxt \triangleq hr' = \text{IF } hr \neq 12 \text{ THEN } hr + 1 \text{ ELSE } 1$$

- HCnxt is an *ordinary mathematical formula*, except that it contains primed and unprimed variables. Such a formula is called an **action**.
- An action formula (HCnxt) can be true or false of a step.
- When an HCnxt step occurs, we can say that action HCnxt is executed.

An Hour Clock – full specification

- The idea is **to specify a system by single formula**, combining asserts that (1) its initial state satisfies HCini, and that (2) each of its steps satisfies HCnxt.
- To express (2) we will use the temporal-logic operator □ (pronounced box).
- The temporal formula □ F asserts that formula F is **always true**. In particular, □ **HCnxt** is the assertion that **HCnxt** is true for every step in the behavior.
- **HCini /\ □ HCnxt** is true of a behavior, if the initial state satisfies **HCini** and every step satisfies **HCnxt**.

An Hour Clock – stuttering steps

- Lets the display shows not only the current hour but also temperature. The state of the clock is described by two variables: **hr**, representing the hour, and **tmp**, representing the temperature.
- The example of behavior of the system is

$$\begin{bmatrix} hr & = & 11 \\ tmp & = & 23.5 \end{bmatrix} \to \begin{bmatrix} hr & = & 12 \\ tmp & = & 23.5 \end{bmatrix} \to \begin{bmatrix} hr & = & 12 \\ tmp & = & 23.4 \end{bmatrix} \to$$

$$\begin{bmatrix} hr & = & 12 \\ tmp & = & 23.3 \end{bmatrix} \to \begin{bmatrix} hr & = & 1 \\ tmp & = & 23.3 \end{bmatrix} \to \cdots$$

- In the second and third steps, **tmp** changes but **hr** remains the same.

An Hour Clock – stuttering steps

- Thus, the formula **HCini /\ □ HCnxt** does not describe the measuring temperature clock behavior.
- A formula that describes it must allow steps that leave **hr** unchanged, i.e. **hr` = hr** steps. These are called **stuttering steps**.
- A specification of the measuring temperature hour clock should allow both **HCnxt** steps and stuttering steps, i.e.

  HCnxt \/ (hr' = hr)

An Hour Clock – stuttering steps

- Lets adopt **HCini /\ □ HCnxt**, we will have
  HCini /\ (□ HCnxt \/ (hr` = hr))

- Or, in TLA syntax we need write
  HCini /\ □[HCnxt]$_{hr}$

- This formula allows **stuttering steps**

$$[hr = 10] \rightarrow [hr = 11] \rightarrow [hr = 11] \rightarrow [hr = 11] \rightarrow \cdots$$

- For example, it will allow us to add the **min** variable to specification of Hour Clock system. It will change from **1..60**, while **hr** remains unchanged.

A Closer Look at the Specification

- A state is defined by assignment of values to variables, as e.g. **[*hr* = 11]**

- A behavior is a (infinite or finite) sequence of states, e.g.:

$$[hr = 11] \rightarrow [hr = 12] \rightarrow [hr = 1] \rightarrow [hr = 2] \rightarrow \cdots$$

- Specification of the Hour clock is a temporal formula HC.

$$HC \triangleq HCini \land \Box[HCnxt]_{hr}$$

- A temporal formula is an assertion about behavior. Behavior satisfies HC iff this formula is true in all states of Hour Clock.

A Closer Look at the Speciation

- Thus hr has a value from 1 through 12 in every state of any behavior satisfying the specification HC.
- Formula HCini asserts that hr has value from 1 through 12, and □HCini asserts that HCini is always true. Or in other words HC implies □HCini for any behavior.
- A temporal formula satisfied by every behavior is called a theorem, so HC => □ HCini is a **theorem.**

$$\text{THEOREM } HC \Rightarrow \Box HCini$$

Comparison of ASCII and TLATEX (HourClock.tla)

```
------------------------ MODULE HourClock ------------------------
EXTENDS Naturals
VARIABLE hr
HCini  ==  hr \in (1 .. 12)
HCnxt  ==  hr' = IF hr # 12 THEN hr + 1 ELSE 1
HC  ==  HCini /\ [][HCnxt]_hr
--------------------------------------------------------------
THEOREM  HC => []HCini
==============================================================
```

MODULE *HourClock*

EXTENDS *Naturals*

VARIABLE *hr*

$HCini \triangleq hr \in (1 \,..\, 12)$

$HCnxt \triangleq hr' = \text{IF } hr \neq 12 \text{ THEN } hr + 1 \text{ ELSE } 1$

$HC \triangleq HCini \land \Box[HCnxt]_{hr}$

THEOREM $HC \Rightarrow \Box HCini$

File of configuration - HourClock.cfg

```
(********************************************************************************)
(* This is a TLC configuration file for testing that HCini is an invariant *)
(* of the specification HC                                                 *)
(********************************************************************************)
```

SPECIFICATION HC

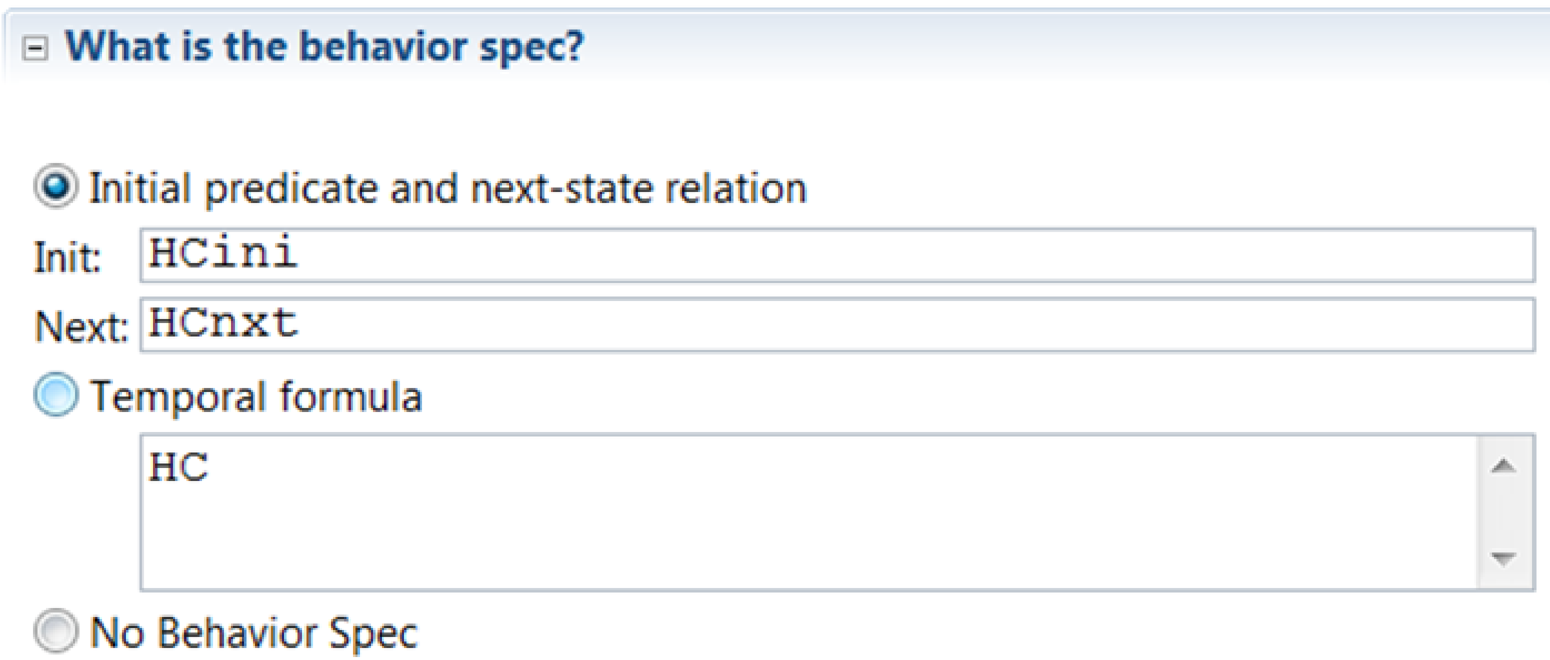
* This statement tells TLC that HC is the specification that it
* should check.

INVARIANT HCini

* This statement tells TLC to check that formula HCini is an
* invariant of the specification--in other words, that the
* specification implies []HCini.

Using TLA ToolBox

TLA ToolBox allows to specify behaviour by initial predicate (HCini) and next state relation (HCnxt).



Using EXTENDS

The specification of the Hour Clock is the definition of formula HC, which using the operators .. and +.

To use them we need include the module **Naturals** by the special keyword **EXTENDS**

TLC asserts that the formula HC follows logically from the definitions in this module, the definitions in the Naturals module, and also the general rules of TLA.

An Alternative Specification

The Naturals module also defines the modulus operator %.
The formula i % n, which mathematicians write i mod n,
is the remainder when i is divided by n.

$$HCnxt2 \triangleq hr' = (hr \% 12) + 1$$

$$HC2 \triangleq HCini \land \Box[HCnxt2]_{hr}$$

Formulas HC and HC2 are equivalent. In other words,
HC <=> HC2 is a theorem.

An Alternative Specification - HourClock2.tla

```
------------------------ MODULE HourClock2 ---------------------------
EXTENDS HourClock

HCnxt2  ==  hr' = (hr % 12) + 1
HC2  ==  HCini /\ [][HCnxt2]_hr
---------------------------------------------------------------------
THEOREM  HC <=> HC2
```

Note, using Extends allows us to make composition of specifications

An Alternative Speciation - HourClock2.cfg

```
(*************************************************************************)
(* This is a TLC configuration file for testing that HC2 implies HC.  *)
(*************************************************************************)

SPECIFICATION HC

\* This statement tells TLC that it is to take formula HC as
\* the specification it is checking.

PROPERTY HC2
\* This statement tells TLC to check that the specification
\* implies the property HC2.
\* (In TLA, a specification is  also a property.)
```

Thank you for your attention!
Please ask questions

Questions for control

1. What is TLA and why TLA is needed?
2. For what kind of systems TLA is good?
3. What is the basic math inside TLA?
4. Explain the implication operation.
5. What is tautology?
6. How we can specify behavior of a system?
7. What is a state of a system?
8. What is a step?
9. What is initial predicate in specification?
10. What is next-state predicate?
11. How to link the previous and the next state of a system?
12. What is an action?
13. What means operator $\Box$ (box)?
14. Explain formula HCini $\wedge$ $\Box$ HCnxt
15. What is stuttering steps? Why we need specify it?
16. Explain formula HCini $\wedge$ $\Box[\text{HCnxt}]_{\text{hr}}$.