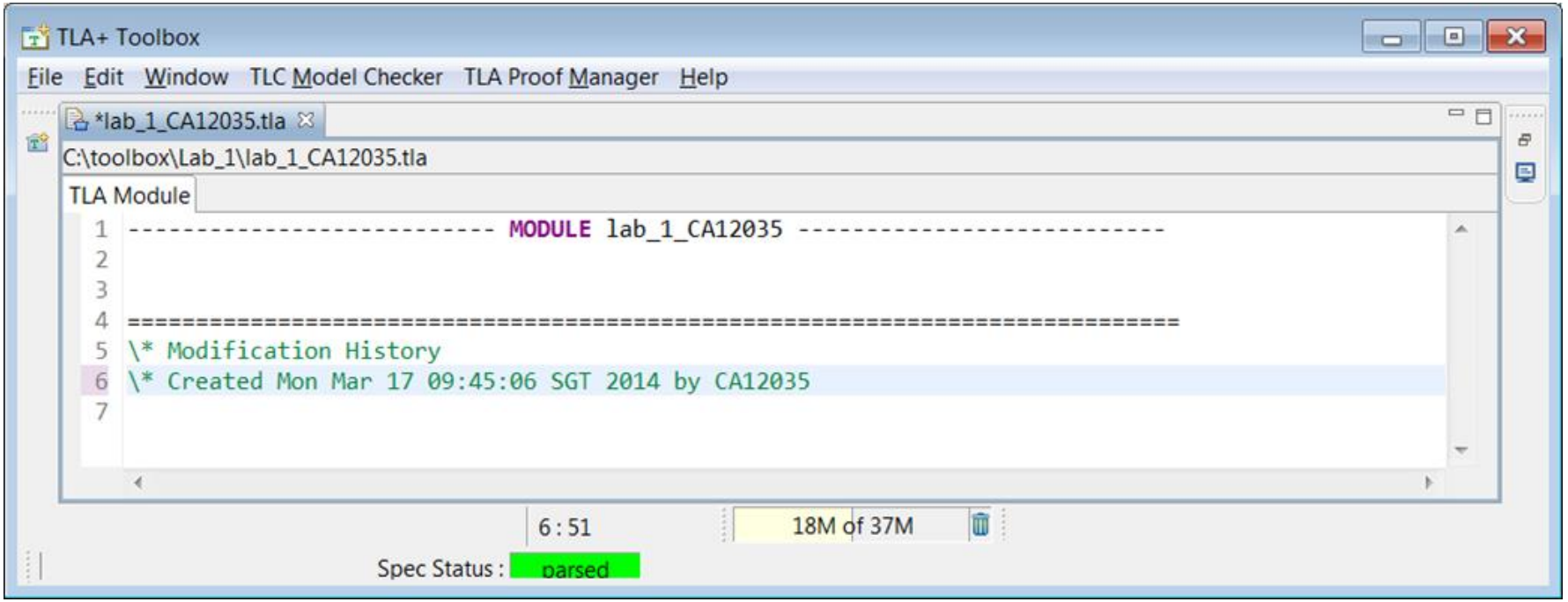
Faculty of Computer Systems & Software Engineering

Formal methods.

Using TLA toolbox and TLC Model Checker

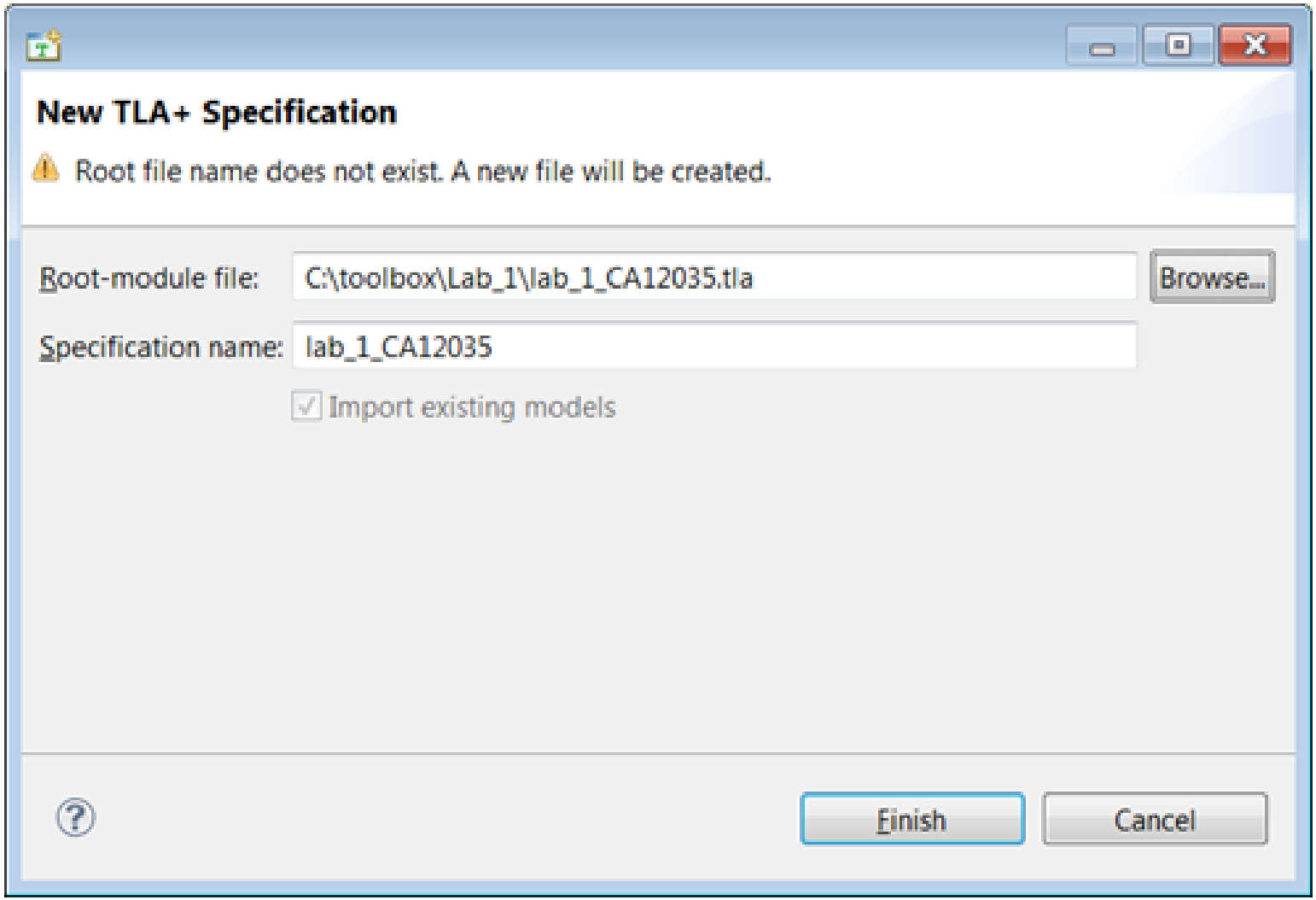
Vitaliy Mezhuyev

Introduction to TLA toolbox

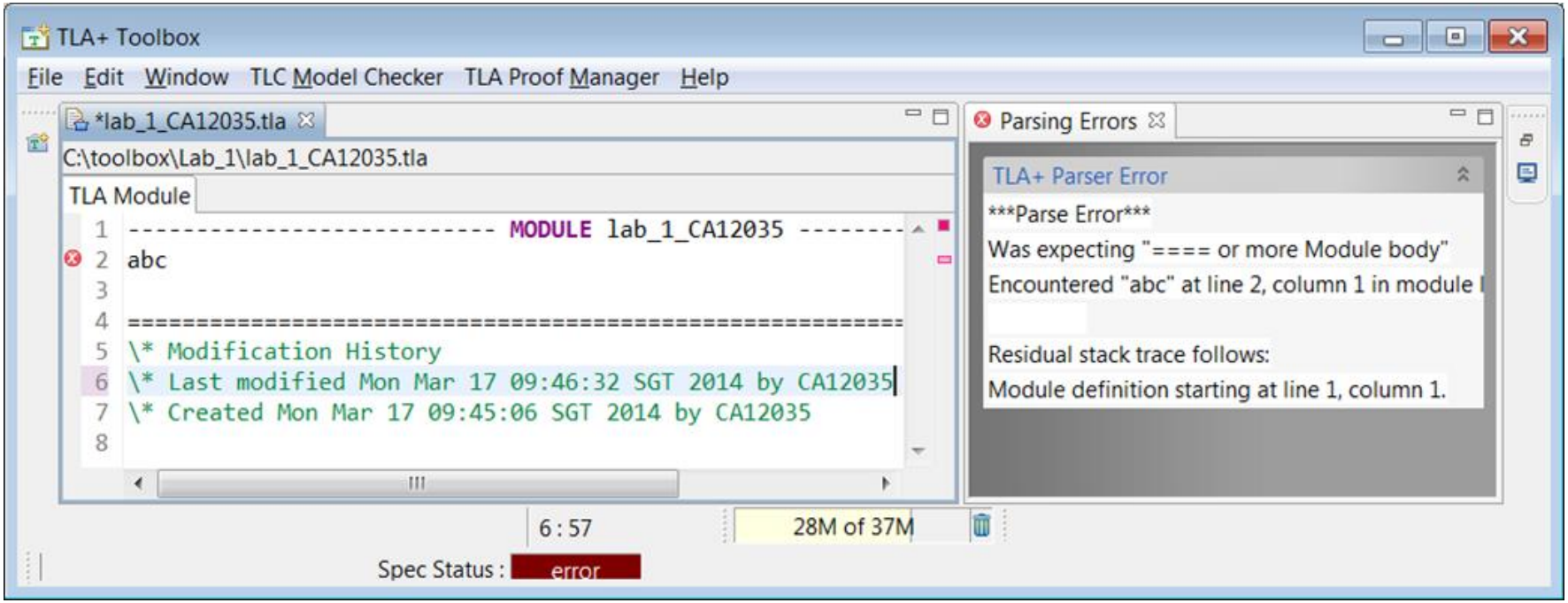


TLA+ Tools: http://research.microsoft.com/en-us/um/people/lamport/tla/tools.html

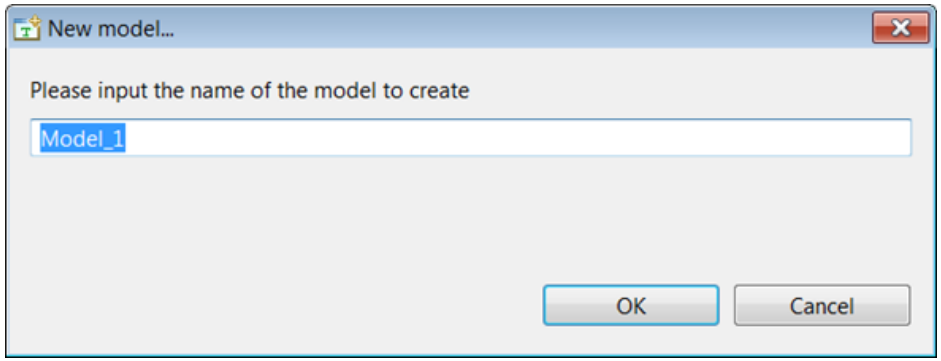
Create new TLA specification



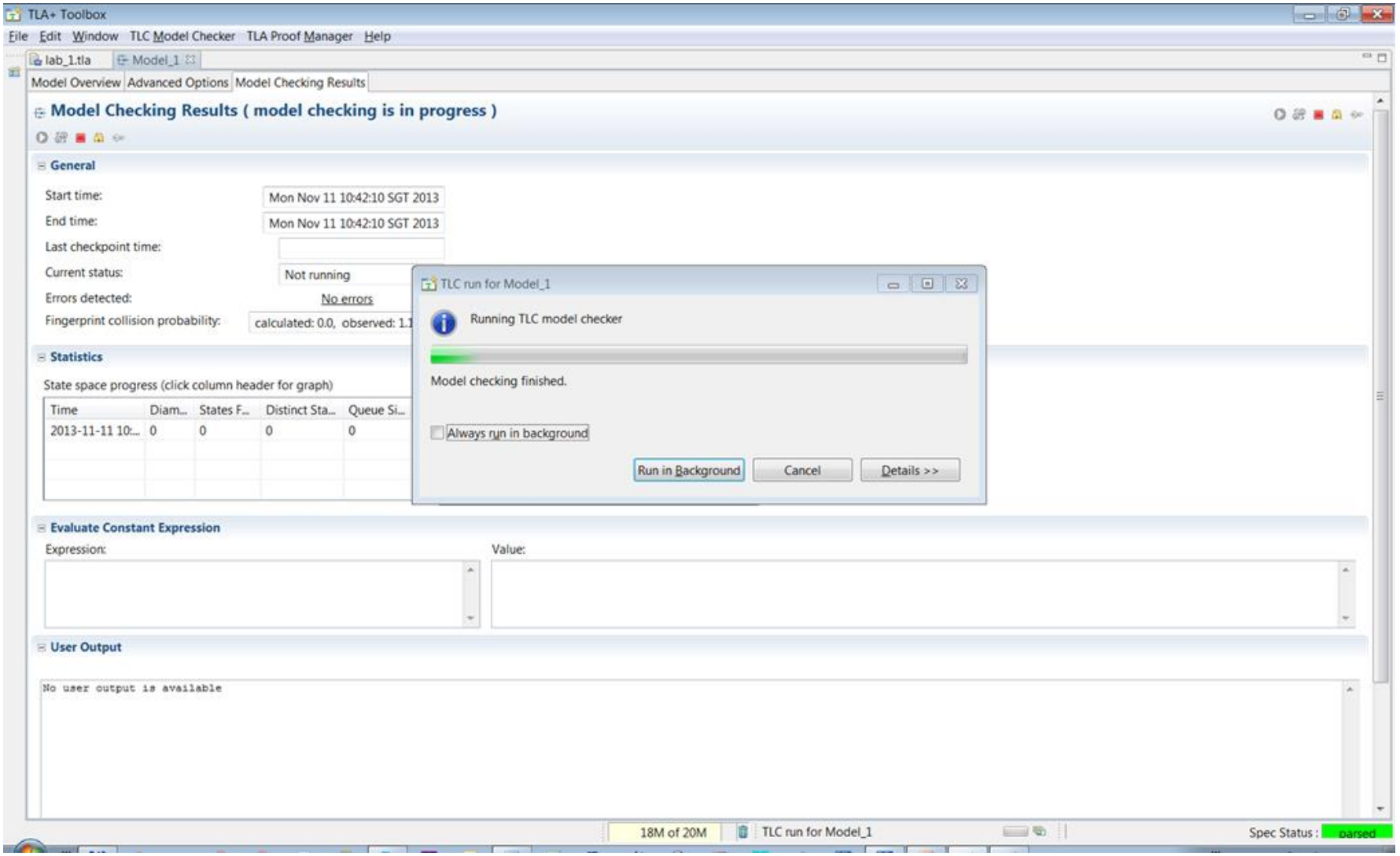
TLA specifications editor



Create a new model (TLC Model Checker -> New model…**)**



Check the model



Introduction to TLC

TLC handles specifications in the standard form

$$Init \wedge \Box[Next]_{vars} \wedge Temporal$$

where
Init is the initial predicate
Next is the next-state action
vars is the *tuple* of all model variables
Temporal is a temporal formula that specifies a *liveness* condition.

Note. Tuple in ASCII is represented by **<< >>**

TLC input

The input to TLC consists of a TLA module and configuration. The configuration tells TLC the names of the specification and of the properties to be checked.

For example for HourClock we need specify behavior as:

Temporal formula
HC

OR

Initial and next state predicates**:**
HCini and HCnext

Using TLA ToolBox

TLC Values

TLA is untyped language.
At definition of variables we do not specify their types.

TLC can compute expressions, built from the following four types of primitive values:

Booleans	Values TRUE and FALSE.
Integers	Values like 123.
Strings	Values like “abc".
Model Values	Values introduced in the CONSTANT statement, e.g. {d1, d2, d3}

Variables in TLC

To check the model of specification TLC generates state space, based on values of variables and constants.

Values of variables TLC deducts from TLA model.

```
VARIABLE hr
HCini  ==  hr \in (1 .. 12)
\* Distinct amount of states is 12

VARIABLE hr, min
HCini  ==  /\ hr \in (1 .. 12)
           /\ min \in (1 .. 60)
\* Distinct amount of states is 12*60 = 720
```

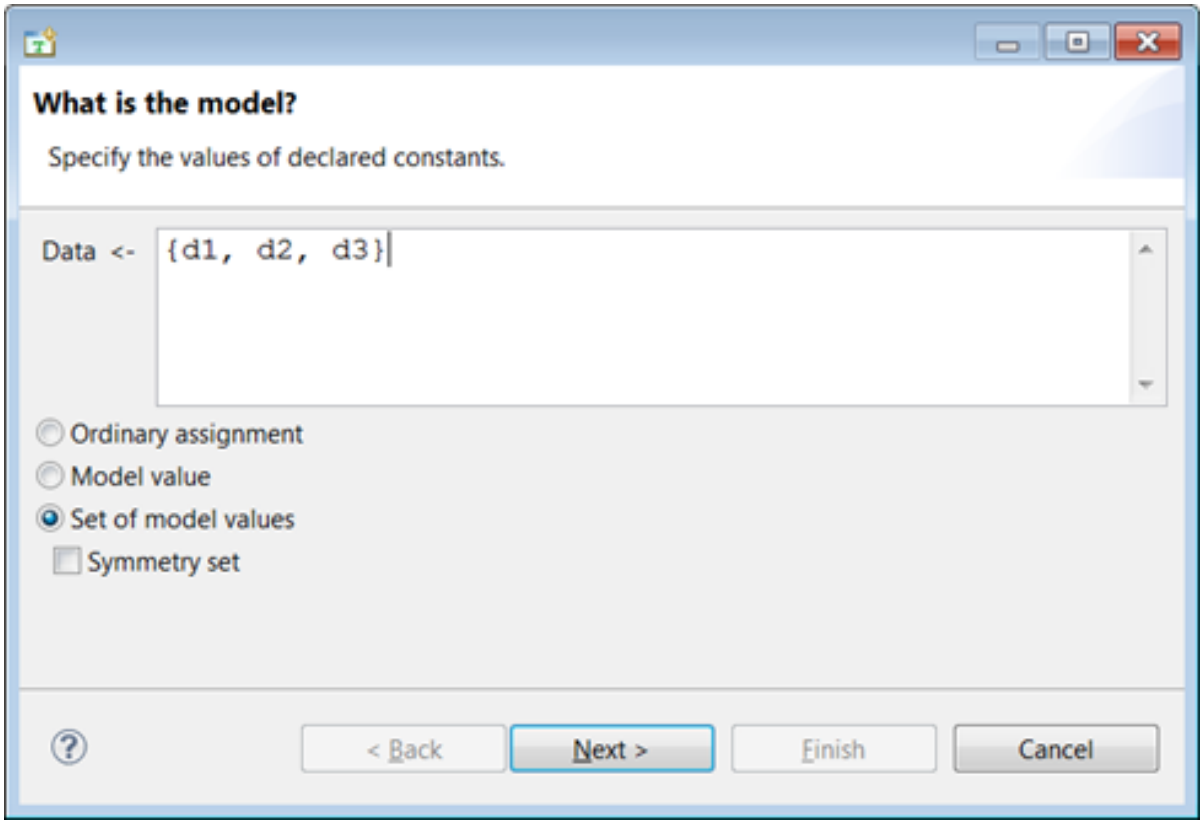
Constants in TLC

To build a model of specification we need *manually* specify values of *constant parameters*.

E.g. in TLA we define some constant
CONSTANT Data

To assign values to the constant Date use the Model Overview page of TLA Toolbox

Model values (Model Overview page)



Modes of TLC

There are two ways to use TLC. The default method is model checking, in which TLC tries to find all reachable states (that is, all states that can occur in behaviors satisfying the formula).

You can also run TLC in simulation mode, in which it randomly generates behaviors, without trying to check all reachable states.

Second way is useful then we have huge amount of states.

Types of errors to be checked

To find errors in a specification is to verify that it satisfies properties that it should.

You can also run TLC without having it check any property, in which case it will just look for two kinds of errors:

- “Silliness" errors. A silly expression is one like “3 + <<1”, “2>”, whose meaning is not determined by TLA
- Deadlock. it is expressed by the invariance property

$$\Box(\text{ENABLED } Next)$$

Properties of the behaviour that TLC can check

Deadlock. A *deadlock* is a state for which the next-state relation allows no successor states. Note, there is special type of a deadlock – termination, that not an error. If you want to specify behaviour that allows termination, then you should uncheck the deadlock option.

Invariants. An invariant is a state predicate that is true in all reachable states. You can include a list of invariants. The checking of each invariant can be enabled or disabled by checking or unchecking its box.

Properties. TLC can check if the behaviour spec satisfies (implies) a temporal property, which is expressed as a temporal-logic formula. You can specify a list of such properties, each with a check-box for enabling or disabling its checking.

Properties to be checked

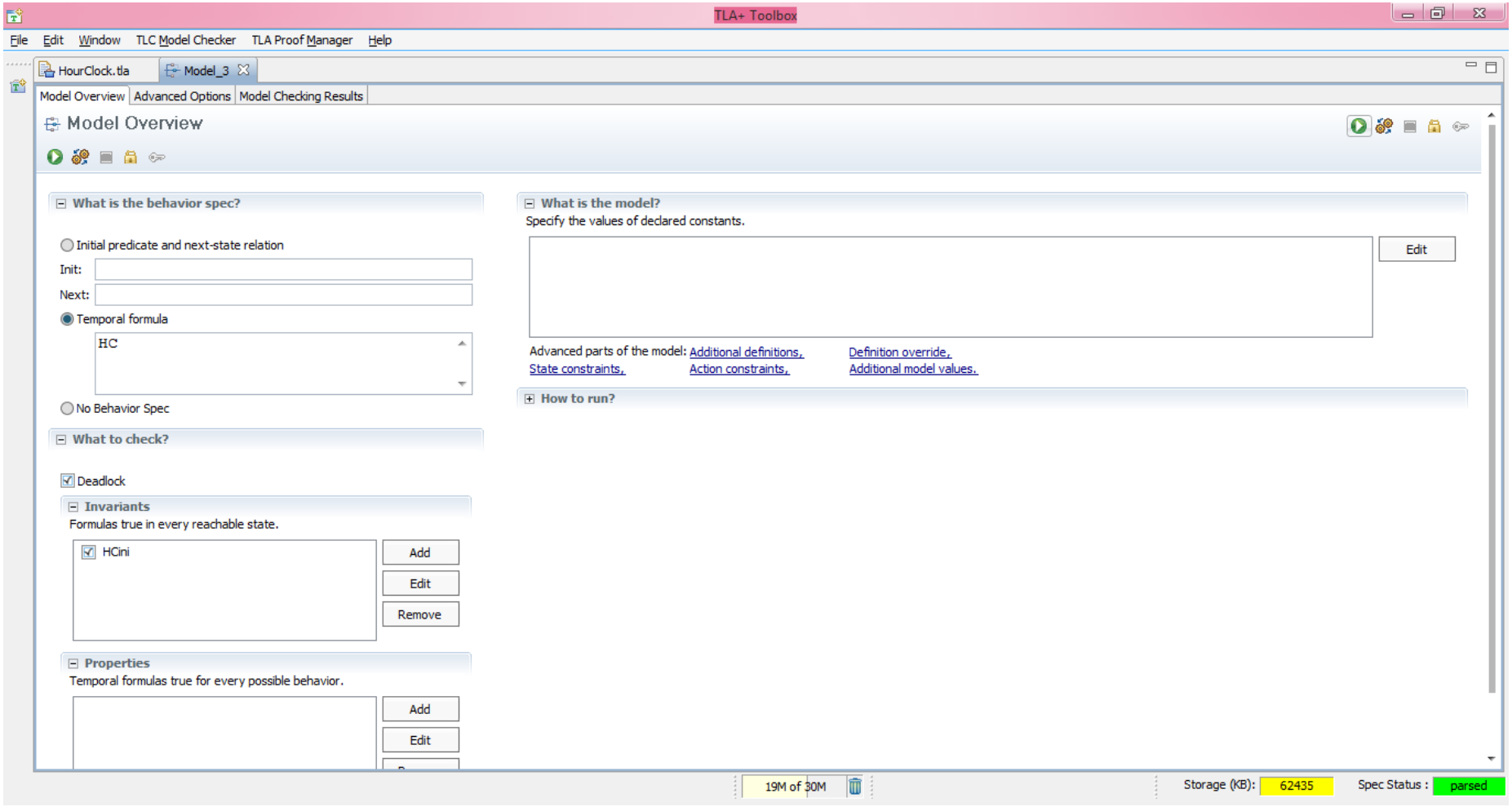
For HourClock.tla TLC checks the **invariant** property **HCini**
Here invariant **HCini** specifies a state predicate.

In other words, TLC checks that formula **HCini** is an invariant of the specification **HC**, or, that the specification implies []HCini.

THEOREM HC => []HCini

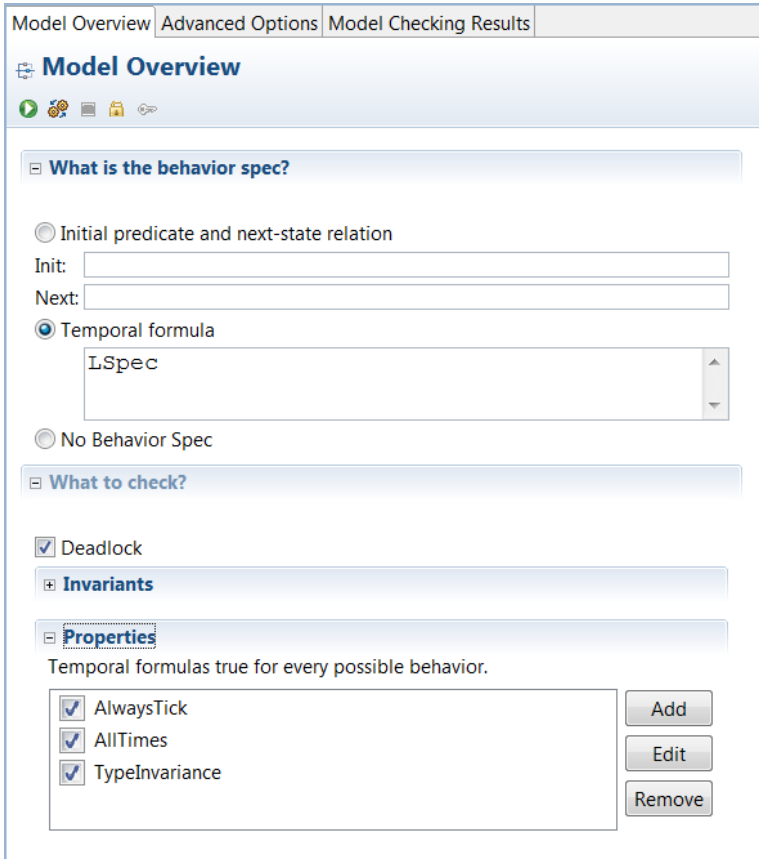
TypeInvariance == []HCini
This formula asserting that HCini is *always* true
THEOREM HC => TypeInvariance

TLA Toolbox – Model Overview page



Properties to be checked

For **LiveHourClock** we will introduce liveness properties
PROPERTIES AlwaysTick, AllTimes, TypeInvariance
THEOREM LSpec => AlwaysTick /\ AllTimes /\ TypeInvariance



How TLC Evaluates Expressions

TLC generally evaluating subexpressions “from left to right". In particular:

- It evaluates $p \land q$ by first evaluating p and, if it equals TRUE, then evaluating q.
- It evaluates $p \lor q$ by first evaluating p and, if it equals FALSE, then evaluating q. It evaluates $p \Rightarrow q$ as $\neg p \lor q$.
- It evaluates IF p THEN e_1 ELSE e_2 by first evaluating p, then evaluating either e_1 or e_2.

Using comments in TLA

- A comment may appear anywhere enclosed between (* and *)
- An end-of-line comment is preceded by *
- Comments may be nested, so you can comment out a section of a commented specification by enclosing it between (* and *)

Using comments in TLA

```
---------------------- MODULE HourClock ----------------------
  (*************************************************)
  (* This module specifies a digital clock that displays  *)
  (* the current hour.  It ignores real time, not         *)
  (* specifying when the display can change.              *)
  (*************************************************)
EXTENDS Naturals
VARIABLE hr     \* Variable hr represents the display.
HCini  == hr \in (1 .. 12)   \* Initially, hr can have any
                             \* value from 1 through 12.
HCnxt  (* This is a weird place for a comment. *)  ==
  (*************************************************)
  (* The value of hr cycles from 1 through 12.     *)
  (*************************************************)
  hr' = IF hr # 12 THEN hr + 1 ELSE 1
HC  ==  HCini /\ [][HCnxt]_hr
  (* The complete spec.  It permits the clock to stop. *)
-----------------------------------------------------------
THEOREM  HC => []HCini  \* Type-correctness of the spec.
===========================================================
```

Thank you for your attention!
Please ask questions