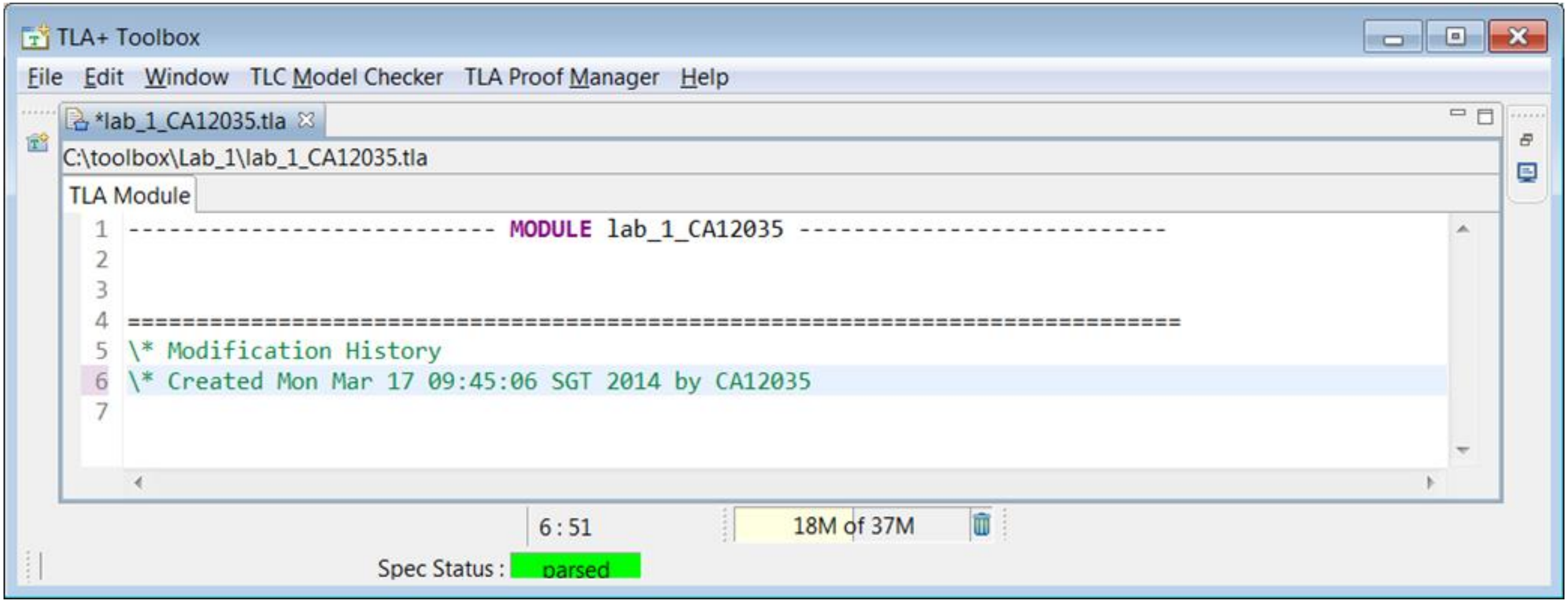
Faculty of Computer Systems & Software Engineering

Formal methods.

Using TLA toolbox and TLC Model Checker

Vitaliy Mezhuyev

Introduction to TLA toolbox



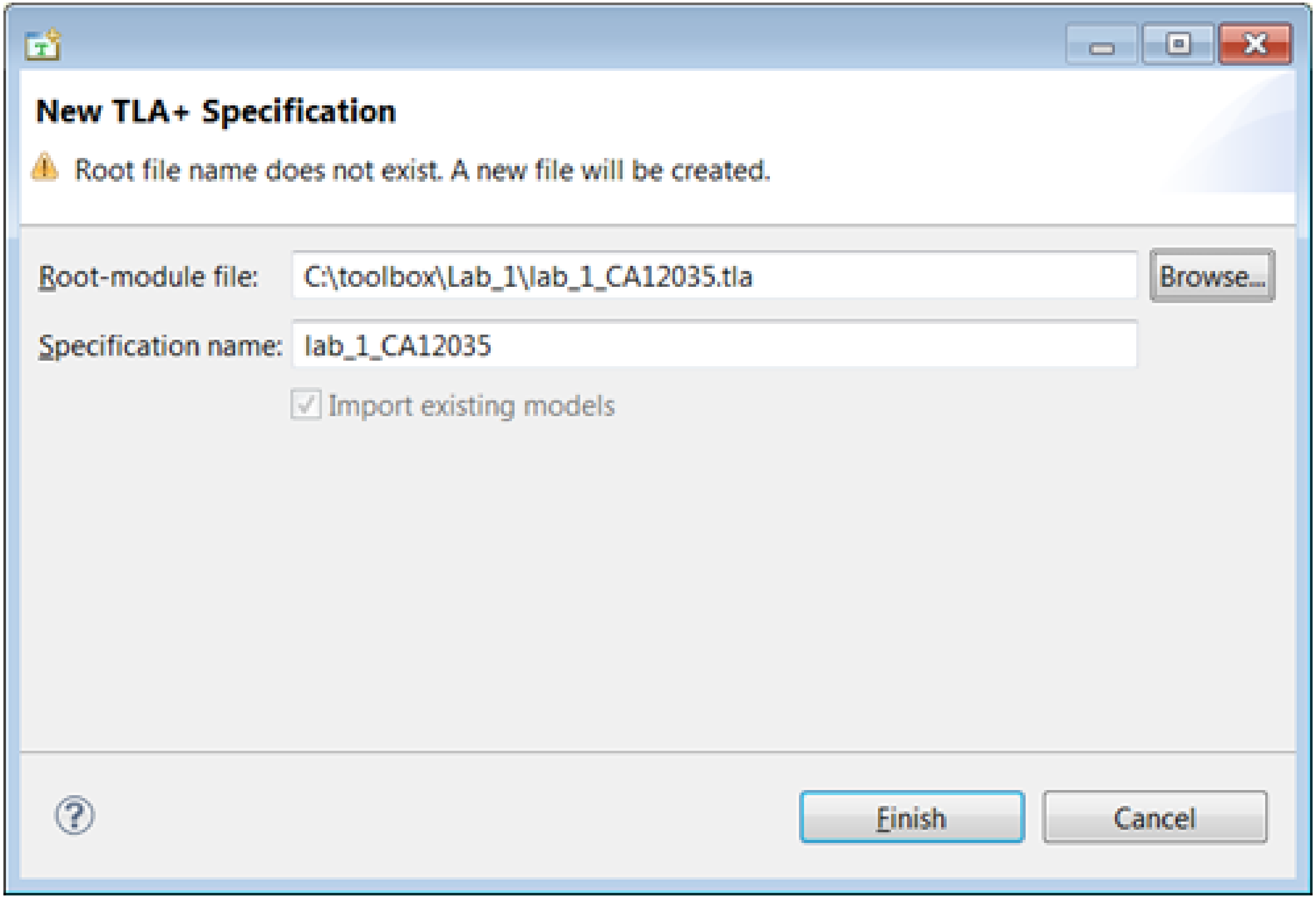
C:\toolbox\Lab_1\lab_1_CA12035.tla

```
--------------------------- MODULE lab_1_CA12035 ---------------------------


=============================================================================
\* Modification History
\* Created Mon Mar 17 09:45:06 SGT 2014 by CA12035

```

Spec Status : parsed

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

Create new TLA specification



New TLA+ Specification

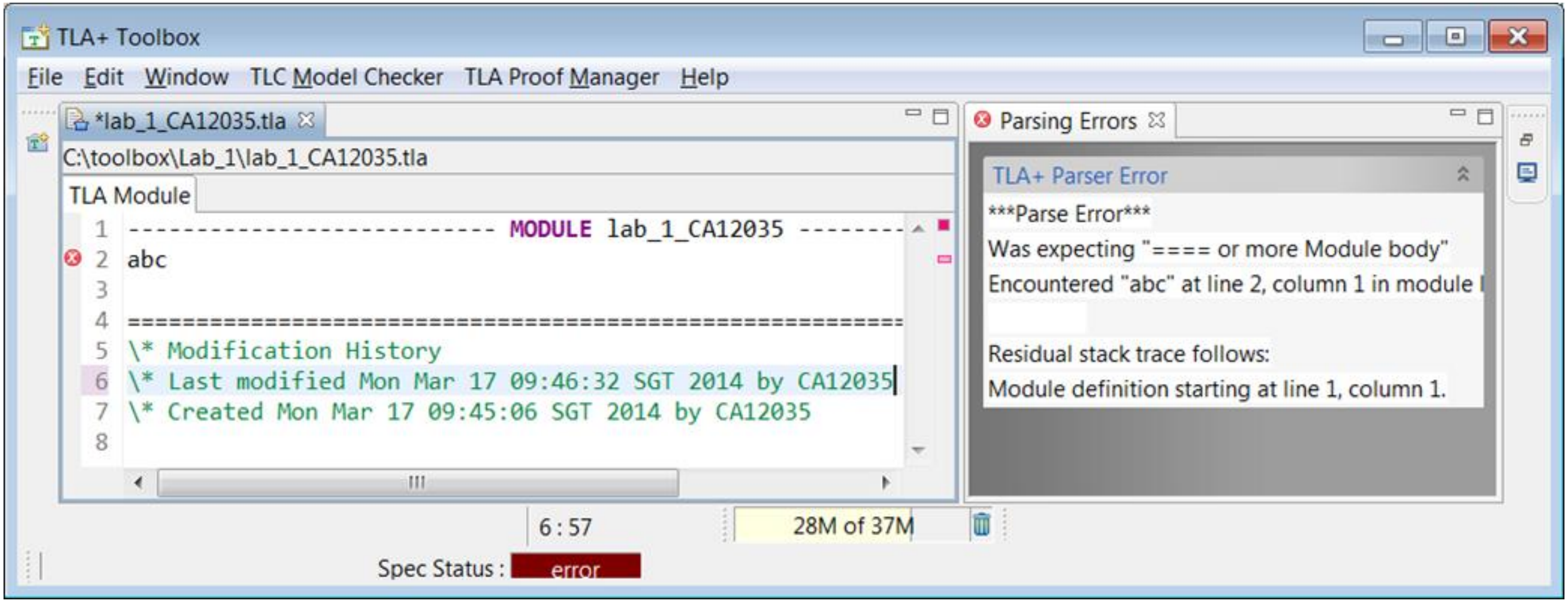
Root file name does not exist. A new file will be created.

Root-module file: C:\toolbox\Lab_1\lab_1_CA12035.tla

Specification name: lab_1_CA12035

☑ Import existing models

TLA specifications editor



C:\toolbox\Lab_1\lab_1_CA12035.tla

```
1  --------------------------- MODULE lab_1_CA12035 --------
2  abc
3
4  =========================================================
5  \* Modification History
6  \* Last modified Mon Mar 17 09:46:32 SGT 2014 by CA12035
7  \* Created Mon Mar 17 09:45:06 SGT 2014 by CA12035
8
```

TLA+ Parser Error

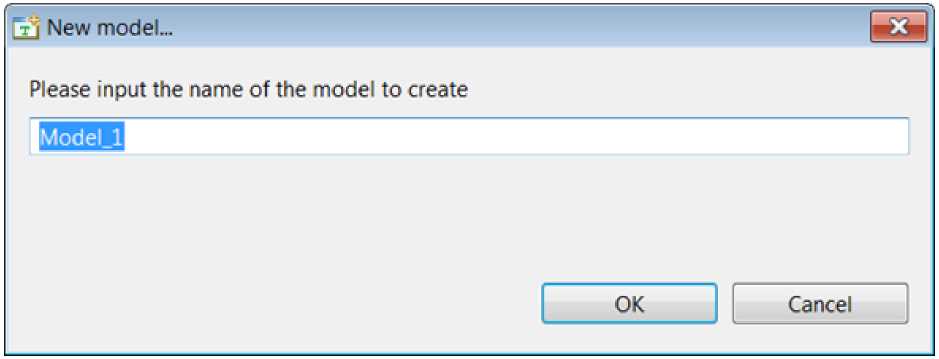
```
***Parse Error***
Was expecting "==== or more Module body"
Encountered "abc" at line 2, column 1 in module l

Residual stack trace follows:
Module definition starting at line 1, column 1.
```

6 : 57

28M of 37M

Spec Status : error

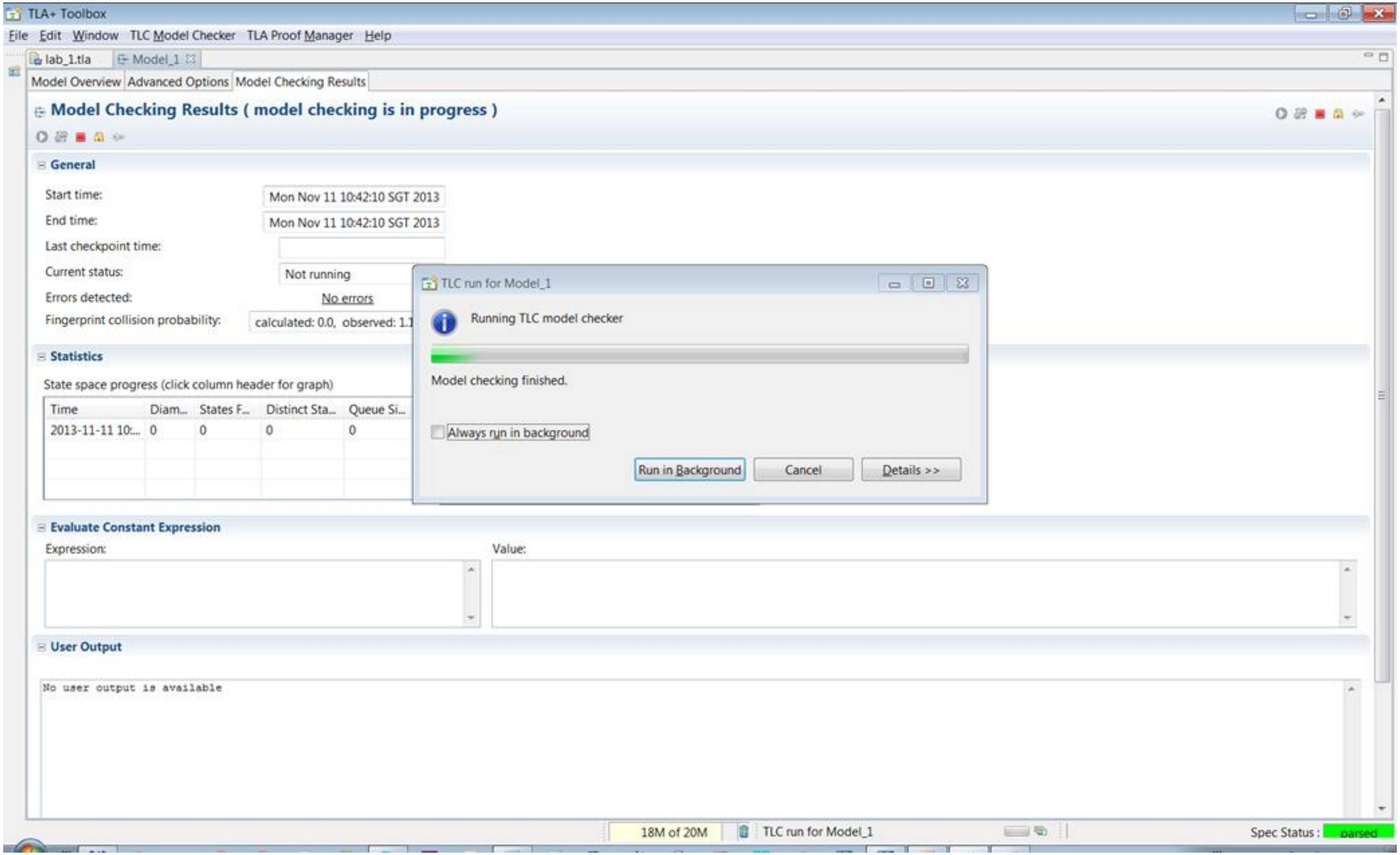
Create a new model (TLC Model Checker -> New model…**)**



Please input the name of the model to create

Model_1

Check the model



Model Checking Results (model checking is in progress)

General

Start time: Mon Nov 11 10:42:10 SGT 2013

End time: Mon Nov 11 10:42:10 SGT 2013

Last checkpoint time:

Current status: Not running

Errors detected: No errors

Fingerprint collision probability: calculated: 0.0, observed: 1.1

Statistics

State space progress (click column header for graph)

Time	Diam...	States F...	Distinct Sta...	Queue Si...
2013-11-11 10:...	0	0	0	0

Evaluate Constant Expression

Expression:

Value:

User Output

```
No user output is available
```

TLC run for Model_1

Running TLC model checker

Model checking finished.

□ Always run in background

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.

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.

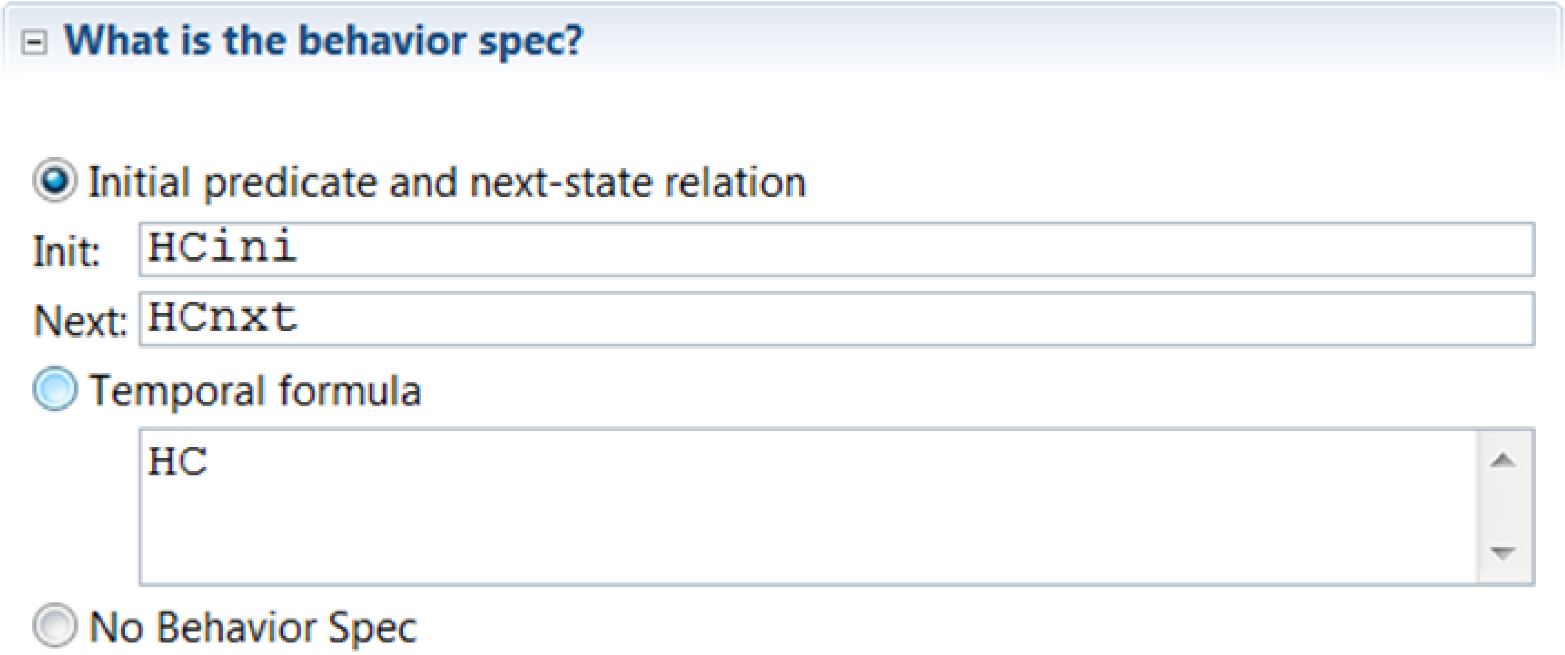
For example for HourClock we need specify behavior as:

SPECIFICATION **HC**
(this statement tells TLC that HC is the specification that it should check)

OR we can specify initial and next state predicates**:**
HCini and HCnext

Using TLA ToolBox

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



⊟ **What is the behavior spec?**

● Initial predicate and next-state relation

Init: HCini

Next: HCnxt

○ Temporal formula

HC

○ No Behavior Spec

TLC Values

TLC can compute a restricted class of values, are 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}

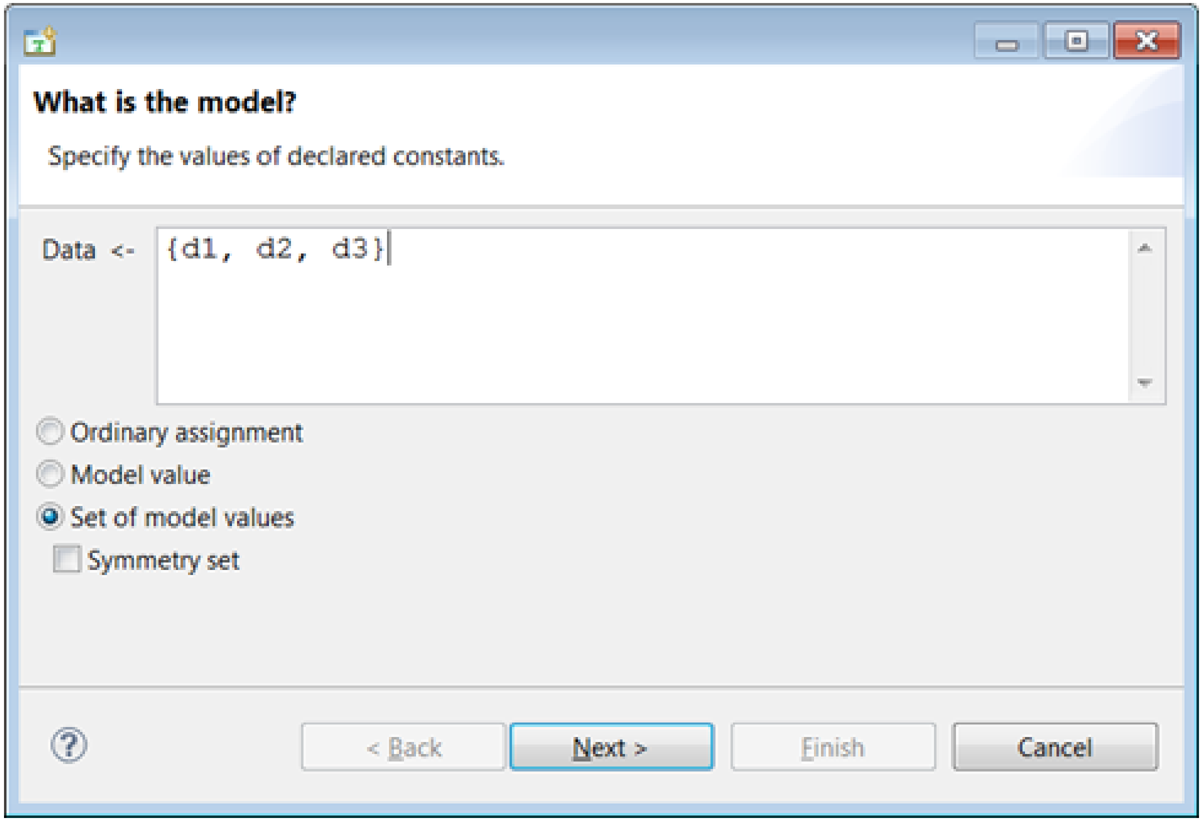
Constants in TLC

TLC works by generating behaviors that satisfy a *model of specification*.

To define a model, we need assign values to the *constant parameters (model values)*.

E.g.
CONSTANT Data = {d1, d2, d3}

Model values (Model Overview page)



What is the model?

Specify the values of declared constants.

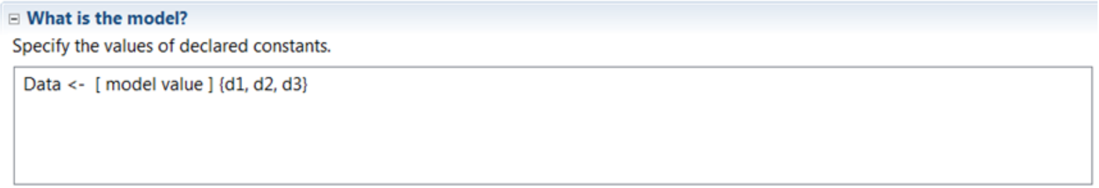
```
Data <- {d1, d2, d3}
```

○ Ordinary assignment

○ Model value

● Set of model values

□ Symmetry set



What is the model?

Specify the values of declared constants.

Data <- [model value] {d1, d2, d3}

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 the semantics of TLA

- Deadlock. it is expressed by the invariance property $\Box(\textsc{enabled}\ \mathit{Next})$

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, termination is a deadlock 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--that is, states that can occur in a behavior allowed by the behavior spec. 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 behavior 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 check 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.

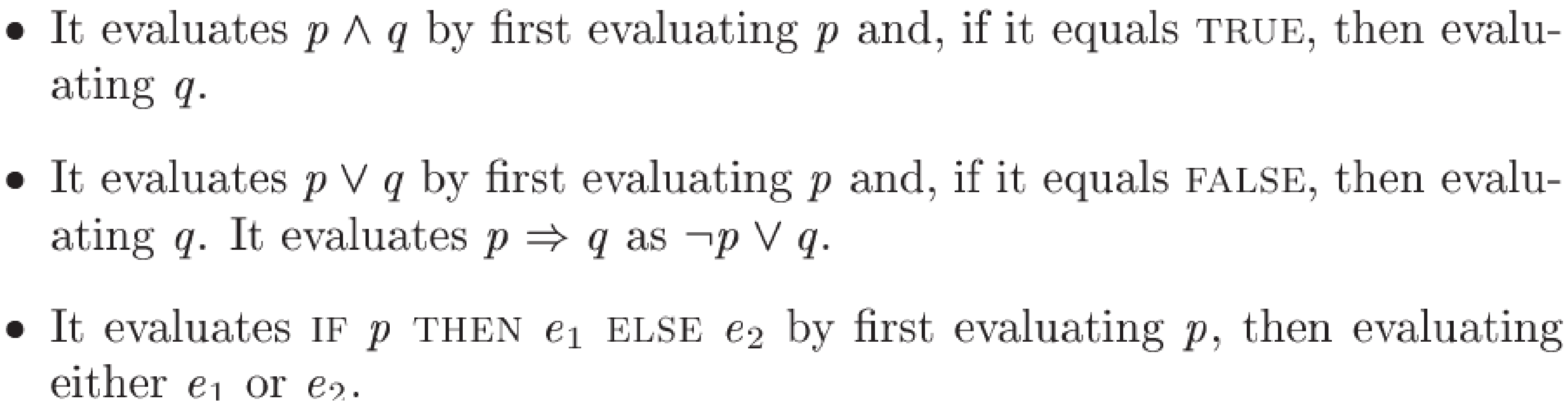
THEOREM HC => []HCini

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

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

How TLC Evaluates Expressions

TLC evaluates expressions in a straightforward way, 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.

Thank you for your attention!
Please ask questions