



TLA Standard

Module format

---- MODULE filename ----	Header of any module
EXTENDS mname	Adds the content of module to the current one
CONSTANTS c1, ...	Defines constant names for the module
VARIABLES v1, ...	Defines global variables for the module
M == INSTANCE mname	Creates a namespace for an imported module
vname == <i>expr</i>	defines a global value
fun(arg1, ...) == <i>expr</i>	Defines a global function
RECURSIVE fun(_,...)	Declares the future definition of a recursive function
fun[x \in S] == <i>expr</i>	Defines a function whose arguments belong to a given set (may be recursive without declaring beforehand)
LOCAL name...	Defines a local value or function
ASSUME <i>P</i>	Asserts <i>P</i> as an assumption
*	Starts a single-line comment
====	Footer of any module

Generic expressions

<i>TRUE</i> , <i>FALSE</i>	Booleans
\/, /\, ~	Bool. operators <i>or</i> , <i>and</i> , <i>not</i>
<< <i>val1</i> , ... >>	Sequences/tuples
[field -> <i>expr</i> , ...]	Records
[r EXCEPT !.field1 = <i>expr</i> , ...]	Record update
r.field	Record access
[x \in S -> <i>expr</i>]	Functions
[f EXCEPT ![x] = <i>expr</i> , ...]	Function update
f[x]	Function call
{x1, x2, ...}	Sets
{ <i>expr</i> : x \in S}	Mapping of set
{x \in S : p}	Filtering of set
DOMAIN <i>f</i>	Domain of the function/tuple/sequence <i>f</i>
LET v == <i>expr</i> IN ...	Local variable
IF <i>expr</i>	Conditional statement
THEN <i>expr</i>	
ELSE <i>expr</i>	

Module/value

CASE $p_1 \rightarrow expr_1$
|| $p_2 \rightarrow expr_2 \dots$
OTHER $\rightarrow expr$

Boolean Predicates

\E x \in S: *p*
 \A x \in S: *p*
CHOOSE x \in S: *p*
 $p \Rightarrow q$
 $p \Leftrightarrow q$

Action Predicates

e'
 $[A]_e$
 $\langle A \rangle_e$
UNCHANGED *e*
ENABLED *A*

Temporal Predicates

|| *p*
 $\langle \rangle p$
 $p \sim q$
WF_{*e*}(*A*)
SF_{*e*}(*A*)

Always *p*
 Eventually *p*
p leads to *q*
 Weak Fairness for action *A*
 Strong Fairness for action *A*

Useful modules

Sequences

Sequences are 1-indexed

$s[i]$
Head(*s*)
Tail(*s*)

Append(*s*, *i*)

\o
Len(*s*)
Seq(*S*)

i^{th} element of the sequence
 First element of a sequence
 The sequence without its head
 Adds *i* at the end of sequence *s*
 Concatenation
 Length of a sequence
 Sequences of elements of set *S*

FiniteSets

x \in S
 x \notin S
S \subseteq *T*
S \union *T*
S \intersect *T*
S \ T
SUBSET *S*
UNION *S*
IsFiniteSet(*S*)
Cardinality(*S*)

x is in set *S*
x is not in set *S*
S is a subset of *T*
 Union of sets
 Intersection of sets
S without elements of *T*
 All the subsets of *S*
 Flatten sets of sets
TRUE if *S* is finite
 Number of elements of *S*

Use the value defined in a namespace

Selects an $expr_i$ such that p_i is *TRUE*, otherwise selects *expr*

Existential quantifier
 Universal quantifier
 Selection in set
 Implication
 Equivalence

The value of *e* after a step
 A or $e' = e$
 A and $e' \neq e$
 e did not change
 A is possible

Always *p*
 Eventually *p*
p leads to *q*
 Weak Fairness for action *A*
 Strong Fairness for action *A*

Naturals, Integers

+, −*, *, \div
 x^y
<, >, \leq, \geq
%
x..y
 * only available in the Integer module

Arithmetical operators
 x to the y
 Comparison operators
 Modulo
 $\{x, x + 1, \dots, y\}$

Reals

Real
 /
Infinity

Set of reals
 Real division
 Value greater than any real (NOT A REAL)

TLC

Print(*msg*, *val*)

Print(*msg*)
Assert(*val*, *out*)

$d := e$
 $f @ g$
SortSeq(*s*, *Op*(_, _))
ToString(*v*)
TLCeval(*v*)

Prints *msg*, then returns *val*
Print(*msg*, *TRUE*)
 Prints *out* and fails iff *val* is *FALSE*
 $f(d) = e$
 Union of functions
 Sorts a sequence
 String representation of *v*
 Forces evaluation of *v*

Bags

A bag is a set that can contain multiple (finite) copies of the same element.

EmptyBag
IsABag(*B*)
BagToSet(*B*)
SetToBag(*S*)
BagIn(*B*, *e*)
 (+), (−)
BagUnion(*S_B*)
 $B1 \sqsubseteq B2$
SubBag(*B*)
BagOfAll(*F*(_), *B*)
BagCardinality(*B*)
CopiesIn(*e*, *B*)

The empty bag
 Checks if *B* is a bag
 The set of bag elements
 The bag of set elements
 Checks if *e* is in the bag
 Union, disjunction
 Union of set of bags
 Subset
 Set of all sub-bags
 Mapping on bags
 Size of a bag
 Number of *e* in the bag *B*

Json

ToJson(*v*)
ToJsonArray(*v*)
ToJsonObject(*v*)
JsonSerialize(*file**, *v*)

ndJsonSerialize(*file**, *v*)

JsonSerialize(*file**)
ndsonSerialize(*file**)

Returns *v* as a Json string
 Same, but for a sequence
 Returns a Json object
 Writes *v* as a (plain) Json in *file*
 Same, but Json is newline delimited
 Returns the content of *file*
 Same, but values must be newline delimited

* file name must be absolute



Creating a model

Counter.tla

Implements a simple counter

```
---- MODULE Counter ----

EXTENDS Naturals

\* An unknown constant
CONSTANTS MAX

\* The variables of our model
VARIABLES counter, reset

\* The initial state (must be finite)
Init ==
  /\ counter \in 0..MAX
  /\ reset \in {TRUE, FALSE}

\* If `reset' is set, then counter is reinitialized
Incr ==
  /\ ~reset
  /\ reset' \in {TRUE, FALSE}
  /\ counter' = counter + 1

\* If `reset' is set, then counter is reinitialized
Reset ==
  /\ reset
  /\ reset' \in {TRUE, FALSE}
  /\ counter' = 0

\* The Next state predicate
Next ==
  \/ Incr
  \/ Reset

\* The specification of our model:
\* - starts by Init
\* - Next is the next state predicate, but variables
\   are allowed not to change between steps
Spec ==
  /\ Init
  /\ [][Next]_<<counter, reset>>
====
```

Props.tla

Properties on the counter model

```
---- MODULE Props ----

CONSTANT MAX

VARIABLES counter, reset

LOCAL INSTANCE Counter WITH
  MAX <- MAX,
  counter <- counter,
  reset <- reset

\* Invariants

\* Variable `counter' is always positive
AlwaysPositive == counter >= 0

\* Temporal Properties

\* If `reset' happens, then `counter' will be 0
ResetLeadsToZero ==
  reset ~> counter = 0

\* Either:
\*   in the future, `reset' will never be triggered;
\*   or `counter' repeatedly reaches 0.
CounterRuns ==
  \/ <>[](~reset)
  \/ []<>(counter = 0)
====
```

Props.cfg

```
CONSTANT
  MAX = 0

SPECIFICATION
  Spec

INVARIANTS
  AlwaysPositive

PROPERTIES
  ResetLeadsToZero
  CounterRuns
```

Output

Temporal properties were violated.

The following behavior constitutes a counter-example:

```
1: <Initial predicate>
   /\ counter = 0
   /\ reset = FALSE
2: <Action line 7, col 1 to line 10, col 16 of module Props>
   /\ counter = 1
   /\ reset = FALSE
3: <Action line 7, col 1 to line 10, col 16 of module Props>
   /\ counter = 2
   /\ reset = FALSE
4: <Action line 7, col 1 to line 10, col 16 of module Props>
   /\ counter = 3
   /\ reset = TRUE
5: Stuttering
```