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 == *expr* — defines a global value
- fun(arg1, ...) == *expr* — Defines a global function
- **RECURSIVE** fun(_,...) — Declares the future definition of a recursive function
- fun[x \in *S*] == *expr* — 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** *P* — Asserts *P* as an assumption
- * — Starts a single-line comment
- ==== — Footer of any module

Generic expressions

- *TRUE*, *FALSE* — Booleans
- \/, /\, ~ — Boolean operators *or*, *and*, *not*
- << *val*1, ... >> — Sequences/tuples
- [field |-> *expr*, ...] — Records
- [r **EXCEPT** !.field1 = *expr*, ...] — Record update
- r.field — Record access
- [x \in *S* |-> *expr*] — Functions
- [f **EXCEPT** ![x] = *expr*, ...] — Function update
- f[x] — Function call
- {x1, x2, ...} — Sets
- {*expr*: x \in *S*} — Mapping of set
- {x \in *S* : p } — Filtering of set
- **DOMAIN** *f* — Domain of the function/tuple/sequence *f*
- **LET** v == *expr* **IN** ... — Local variable
- Module**!**value — Use the value defined in a namespace
- **IF** *expr* **THEN** *expr* **ELSE** *expr* — Conditional statement
- **CASE** p_1 -> $expr_1$ [] p_2 -> $expr_2$ ... **OTHER** -> *expr* — Selects an $expr_i$ such that p_i is *TRUE*, otherwise selects *expr*

Boolean Predicates

- \E *x* \in *S*: *p* — Existential quantifier
- \A *x* \in *S*: *p* — Universal quantifier
- **CHOOSE** x \in *S*: p — Selection in set
- *p* => *q* — Implication
- *p* <=> *q* — Equivalence

Action Predicates

- e' — The value of e after a step
- $[A]_e$ — A or $e' = e$
- $< A >_e$ — A and $e' \neq e$
- **UNCHANGED** *e* — *e* did not change
- **ENABLED** *A* — *A* is possible

Temporal Predicates

- [] *p* — Always *p*
- <> *p* — Eventually *p*
- *p* ~> *q* — *p* leads to *q*
- $\mathbf{WF}_e(A)$ — Weak Fairness for action *A*
- $\mathbf{SF}_e(A)$ — Strong Fairness for action *A*

Useful modules

Sequences

Sequences are 1-indexed

- *s*[*i*] — i^{th} element of the sequence
- **Head**(*s*) — First element of a sequence
- **Tail**(*s*) — The sequence without its head
- **Append**(*s*, *i*) — Adds *i* at the end of sequence *s*
- s_1 \o s_2 — Concatenation
- **Len**(*s*) — Length of a sequence
- **Seq**(*S*) — Sequences of elements of set *S*

FiniteSets

- *x* \in *S* — *x* is in set *S*
- *x* \notin *S* — *x* is not in set *S*
- *S* \subseteq *T* — *S* is a subset of *T*
- *S* \union *T* — Union of sets
- *S* \intersect *T* — Intersection of sets
- *S* \ *T* — *S* without elements of *T*
- **SUBSET** *S* — All the subsets of *S*
- **UNION** *S* — Flatten sets of sets
- **IsFiniteSet**(*S*) — *TRUE* if *S* if finite
- **Cardinality**(*S*) — Number of elements of *S*

Naturals, Integers

- **Nat**, **Int*** — Sets of numbers
- +, −*, ∗, \div — Arithmetical operators
- *x*^*y* — *x* to the *y*
- <, >, \leq, \geq — Comparison operators
- % — Modulo
- *x*..*y* — $\{x, x+1, ..., y\}$

* only available in the Integer module

Reals

- **Real** — Set of reals
- / — Real division
- **Infinity** — Value greater than any real (NOT A REAL)

TLC

- **Print**(*msg*, *val*) — Prints *msg*, then returns *val*
- **PrintT**(*msg*) — **Print**(*msg*, *TRUE*)
- **Assert**(*val*, *out*) — Prints *out* and fails iff *val* is *FALSE*
- *cst* **:>** *e* — [*x* \in {*cst*} |-> *e*]
- *f* **@@** *g* — Union of functions
- **SortSeq**(*s*, *Op*(_, _)) — Sorts a sequence
- **ToString**(*v*) — String representation of *v*
- **TLCEval**(*v*) — Forces evaluation of *v*

Bags

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

- **EmptyBag** — The empty bag
- **IsABag**(*B*) — Checks if *B* is a bag
- **BagToSet**(*B*) — The set of bag elements
- **SetToBag**(*S*) — The bag of set elements
- **BagIn**(*B*, *e*) — Checks if *e* is in the bag
- (+), (−) — Union, disjunction
- **BagUnion**(S_B) — Union of set of bags
- *B*1 \sqsubseteq *B*2 — Subset
- **SubBag**(*B*) — Set of all sub-bags
- **BagOfAll**(*F*(_), *B*) — Mapping on bags
- **BagCardinality**(*B*) — Size of a bag
- **CopiesIn**(*e*, *B*) — Number of *e* in the bag *B*

Json

- **ToJson**(*v*) — Returns *v* as a Json string
- **ToJsonArray**(*v*) — Same, but for a sequence
- **ToJsonObject**(*v*) — Returns a Json object
- **JsonSerialize**(*file**, *v*) — Writes *v* as a (plain) Json in *file*
- **ndJsonSerialize**(*file**, *v*) — Same, but Json is newline delimited
- **JsonSerialize**(*file**) — Returns the content of *file*
- **ndsonSerialize**(*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' repeatedely 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
```