# TLA$^+$ Operators Not Shown in Video

Leslie Lamport

19 August 2019

## Boolean Constants

In addition to the Boolean value FALSE, the built-in TLA$^+$ constants include the Boolean value TRUE and the set BOOLEAN that equals the set $\{\text{TRUE}, \text{FALSE}\}$ of Boolean values.

## Functions

The operator DOMAIN is defined so that the domain of a function $f$ equals DOMAIN $f$. Thus, any function $f$ equals $[x \in \text{DOMAIN}\, f \mapsto f[x]]$.

## Quantifiers

There are two useful abbreviations for nesting the quantifiers $\forall$ (`\A`) and $\exists$ (`\E`):

$$\forall\, x \in S, \forall\, y \in S\ : \qquad \text{can be written} \qquad \forall\, x, y \in S$$

and

$$\forall\, x \in S\ :\ \forall\, y \in T\ : \qquad \text{can be written} \qquad \forall\, x \in S, y \in T\ :$$

if the identifier $x$ does not occur in the expression $T$. For example,

$$\forall\, x \in S\ :\ \forall\, y \in T\ :\ \forall\, z \in T\ :\ \forall\, p \in U\ :\ \forall\, q \in V\ :$$

can be written as

$$\forall\, x \in S, y, z \in T, p \in U, q \in V\ :$$

if none of the identifiers $x$, $y$, $z$, $p$, and $q$ appear in any of the expressions $S$, $T$, $U$, or $V$.

## Sets

The set construction operator $\{e : x \in S\}$ can be generalized to expressions like

$$\{e : x, y \in S, z \in T\}$$

The syntax and restrictions on what can follow the ":" are the same as for what can follow $\forall$ or $\exists$ in abbreviations of nested quantifiers.

If $S$ is a set whose elements are sets, then UNION $S$ equals the union of all the sets in $S$—in other words, the set of all elements that are in some element of $S$. For example,

$$\textsc{union}\ \{1\,..\,3,\ \{0, 5\},\ 2\,..\,7,\ \{6, 8\}\}$$

equals

$$1\,..\,3\ \cup\ \{0, 5\}\ \cup\ 2\,..\,7\ \cup\ \{6, 8\}$$

which equals $0\,..\,8$. The UNION and SUBSET operators can be confusing; I sometimes get confused when I use them. You should convince yourself that the following relations are true for any set $S$.[1]

$$\textsc{union}\ (\textsc{subset}\ S)\ = S$$
$$S\ \subseteq\ \textsc{subset}\ (\textsc{union}\ S)$$

## The CASE Construct

The TLA$^+$ CASE construct is almost, but not quite, completely unlike the C **switch** statement. It has these two forms:

$$\begin{array}{rlcl} \textsc{case} & P_1 & \rightarrow & e_1 \\ \Box & P_2 & \rightarrow & e_2 \\ & & \vdots & \\ \Box & P_n & \rightarrow & e_n \end{array} \qquad\qquad \begin{array}{rlcl} \textsc{case} & P_1 & \rightarrow & e_1 \\ \Box & P_2 & \rightarrow & e_2 \\ & & \vdots & \\ \Box & P_n & \rightarrow & e_n \\ \Box & \textsc{other} & \rightarrow & f \end{array}$$

(As usual, $\Box$ and $\rightarrow$ are typed as `[]` and `->`.) If at least one of the $P_i$ equals TRUE, then each of these CASE expressions equals some $e_i$ such that $P_i$ equals TRUE. If $P_i$ equals TRUE for more than one $i$, then which of

[1]Remember that, in TLA$^+$, every value is a set. Therefore, every element of a set is a set, so every set is a set of sets. For example, each of the three elements of $1\,..\,3$ is a set; we just don't know what the elements of any of those three sets are.

those $e_i$ the CASE expression equals is unspecified. (Thus, the order of the $n$ clauses $P_i \rightarrow e_i$ makes no difference.) If none of the $P_i$ equals TRUE, then the second CASE expression equals $f$, while the value of the first CASE expression is unspecified. In this case, TLC reports an error if it tries to evaluate the first CASE expression.

### Constant Declarations

Constant operators that take arguments can be declared, as in:

CONSTANT $Op(\_, \_)$ typed as `CONSTANT Op(_, _)`

If you create a model for running TLC on a spec containing this declaration, you will be able to tell the Toolbox that you want to let $Op$ be an operator such that, for example, $Op(a, b)$ equals $a + 2 * b$ for any values $a$ and $b$.