1 An Abstract View of TLC

TLC checks a model of a specification. A model defines a countably infinite set $\mathcal{V}$ of TLC values. (The set $\mathcal{V}$ can differ for different models because it depends on the set of model values introduced by the model.) The specification declares a set of specification variables. A model state is an assignment of a value in $\mathcal{V}$ to each of those variables. Let $\mathcal{S}$ be the set of all model states.

TLC evaluates a TLA$^+$ state predicate on a state. Let $s \models I.$ be the (truth) value of a state predicate I on state s. TLC evaluates a TLA$^+$ action on a pair of states. Let $\langle s, t \rangle \models A$ the value of action A on state pair $\langle s, t \rangle$.

The model declares a specification of the form

$$Spec \;\triangleq\; Init \;\wedge\; \Box[Next]_{vars} \;\wedge\; Fair$$

and declares certain properties of the specification for TLC to check by trying to find a counter-example. Define

$$SSpec \;\triangleq\; Init \;\wedge\; \Box[Next]_{vars}$$

For a safety property *Prop*, TLC tries to find a behavior (an infinite sequence of states) satisfying *SSpec* that satisfies $\neg Prop$. For a liveness property *Prop*, it tries to find a behavior of *SSpec* that satisfies $Fair \wedge \neg Prop$.

The maximal state predicates and actions in *Fair* and in all the properties that TLC is to check are called the model's *atoms*. Let a *state label* be an assignment of truth values to all the model's state-predicate atoms, and let an *action label* be an assignment of truth values to all the model's action atoms. For any state s, let $\lambda(s)$ be the state label that assigns to each state-predicate atom I the value $s \models I$; and let $\lambda(s, t)$ be the action label that assigns to each action atom A the value $\langle s, t \rangle \models A$.

1.1 The State Graph

Let *StConst* and *ActConst* be the model's state and action constraints—specified by the State Constraint and Action Constraint sections of the models *Advanced Options* page, with default values TRUE. A graph whose nodes are states in $\mathcal{S}$ and edges are pairs of states is called a *state graph* for the model iff it satisfies the following conditions:

1. A state s is in the graph only if $s \models StConst$.
2. An edge $\langle s, t \rangle$ is in the graph only if $\langle s, t \rangle \models Next \wedge ActConst$ or $s = t$.[1]
3. Each state in the graph is reachable from an initial state, which is a state satisfying $s \models Init$.

[1] Note: TLC seems to add self-loops even if *ActConst* implies $var' \neq var$.

The *complete* state graph is the maximal state graph, which may be finite or infinite. The nodes in the complete state graph are the *reachable* states of the model. Define the *depth* of a reachable state s to be the length of the shortest path in the complete state graph from an initial state to s.

A behavior of the model is an infinite path in the complete state graph. We can determine if a behavior is a counterexample to a property by knowing the labels of each node and edge of the behavior; we don't need to know the states represented by the nodes.

A state graph is any subgraph of the complete state graph that satisfies the third condition. A node s of a state graph $\mathcal{G}$ is a *sink* iff $\mathcal{G}$ contains no edge $\langle s, t \rangle$ with $s \neq t$.

A state *tree* is a state graph that is a tree, having a single initial state as its root. A state *forest* is a collection of disjoint (sharing no nodes) state trees.

1.2 Symmetry Reduction

Symmetry reduction uses a group *Sym* of automorphisms of the set $\mathcal{S}$ of model states. (An automorphism of a set is a bijection from the set to itself, and *Sym* a group of automorphisms means that it contains the identity automorphism and for any P in *Sym*, the inverse autmorphism P^{-1} is also in *Sym*.) Latter I'll discuss how *Sym* is specified by the model.

Because *Sym* is a group, it defines an equivalence relation $\approx$ on $\mathcal{S}$ by

$$s \approx t \;\triangleq\; \exists P \in \mathit{Sym} : P(s) = t$$

The quotient set $\mathcal{S}/\approx$ of all equivalence classes of states partitions $\mathcal{S}$. Let $[\![s]\!]$ be the equivalence class of s:

$$[\![s]\!] \;\triangleq\; \{t \in \mathcal{S} : t \approx s\}$$

Define a state graph $\mathcal{G}$ to be irredundant iff for any nodes (states) s and t of $\mathcal{G}$, if $s \neq t$ then $s \not\approx t$. For an irredundant state graph $\mathcal{G}$, we define the $[\![\mathcal{G}]\!]$ to be the smallest graph whose nodes are elements of $\mathcal{S}/\approx$ (equivalence classes of states) labeled with state labels and whose edges are labeled with action labels, such that:

- For each state s of $\mathcal{G}$, $[\![\mathcal{G}]\!]$ contains the node $[\![s]\!]$ labeled with $\lambda(s)$ and the edge from $[\![s]\!]$ to itself labeled $\lambda(s, s)$. The node $[\![s]\!]$ is an *initial* node iff s is an initial state.

- For each non-sink node s in $\mathcal{G}$ and each state t with $[\![t]\!]$ a node of $[\![\mathcal{G}]\!]$ and $\langle s, t \rangle$ an edge of the complete state graph, $\mathcal{G}$ contains an edge from $[\![s]\!]$ to $[\![t]\!]$ labeled with $\lambda(s, t)$.

Note that for any states s and t in $\mathcal{G}$ there can be multiple edges from $[\![s]\!]$ to $[\![t]\!]$, each having a different label.

If π is a path s_1 , s_2, ... in the complete state graph, define $[\![\pi]\!]$ to be the labeled path with nodes $[\![s_1]\!]$, $[\![s_2]\!]$, ... such that each state $[\![s_i]\!]$ is labeled with $\lambda(s_i)$ and each edge $\langle [\![s_i]\!], [\![s_{i+1}]\!] \rangle$ is labeled with $\lambda(s_i, s_{i+1})$.

1.3 Symmetric Models

A state predicate I is *symmetric* iff $s \models I$ equals $P(s) \models I$ for all s in $\mathcal{S}$ and P in *Sym*. An action A is *symmetric* iff $\langle s, t \rangle \models A$ equals $\langle P(s), P(t) \rangle \models A$ for all s, t in $\mathcal{S}$ and P in *Sym*. We say that a model is symmetric iff *Init*, *Next*, UNCHANGED *vars*, *StConst*, *ActConst*, and all its atoms are symmetric.

Theorem 1 *Let $\mathcal{G}$ be an irredundant state graph and Π a (labeled) path in $[\![\mathcal{G}]\!]$ starting at an initial node. If the model is symmetric, then there exists a path π in the complete state graph starting at an initial node such that $\Pi = [\![\pi]\!]$. If Π has a finite number of distinct nodes and Sym is a finite group, then π has a finite number of distinct nodes.*

PROOF: We construct the graph π equal to s_1, s_2, ... inductively as follows.

- Let s_1 be the initial state for which $[\![s_1]\!]$ is the first node of Π. State s_1 exists by definition of an initial node of $[\![\mathcal{G}]\!]$.
- Assume we have constructed the required states s_1, ..., s_n. Let s be the state in $\mathcal{G}$ such that $[\![s_n]\!] = [\![s]\!]$. By definition of $[\![\mathcal{G}]\!]$, there exists a state t such that $\langle s, t \rangle$ is an edge in the complete state graph and the edge of Π from $[\![s_n]\!]$ leads to $[\![t]\!]$ and has the label $\lambda(s, t)$. Choose P in *Sym* so that $s_n = P(s)$ and let $s_{n+1} = P(t)$. Symmetry implies that $\langle s_n, s_{n+1} \rangle$, which equals $\langle P(s), P(t) \rangle$, is an edge in the complete state graph with label $\lambda(s, t)$. From $s_{n+1} = P(t)$ we have $[\![s_{n+1}]\!]$ equals $[\![t]\!]$, which is the next node of Π. Symmetry implies that the label of that node equals $\lambda(s_{n+1})$.

By induction, we see that $[\![\pi]\!]$ equals Π. If Π has a finite number of distinct nodes, then it must pass through some node infinitely many times. That node must equal $[\![s_n]\!]$ for infinitely many values of n. If *Sym* is finite, then there are only a finite number of distinct states in the equivalence class $[\![s_n]\!]$, so the path π must end by looping through finitely many nodes.

YYYYYYYYYYYYYYYY

A model specifies a

Throughout this section, we assume a given model of a specification. The model specifies values overriding of some definitions and the substitution of expressions for all the declared CONSTANTS. These overridings and substitutions may introduce A TLC model can introduce model values into expressions. I assume that these overridings and substitutions have been performed in all expressions. Therefore, declared constants no longer appear anywhere, but expressions may contain model values.

1.4 TLC Values

The semantics of TLA$^+$ is based on ZF set theory. It assumes that every expression written in terms of the primitive TLA$^+$ operators with no free identifiers has a *value* that is a set. (Here, we consider model values and declared constants and variables to be free identifiers). Let an *expressible* value be one that is the value of such an expression.

Two arbitrary expressions are equal if they have the same value for any values of their free identifiers such that:

- A model value does not equal any expressible value.[2]
- An untyped model value is unequal to any other model value.
- A typed model value is unequal to any other model value of the same type.

For any expressions e_1 and e_2, we let $e_1 \approx e_2$ mean that the semantics of TLA$^+$ implies $e_1 = e_2$; and we let $e_1 \not\approx e_1$ mean that the semantics of TLA$^+$ implies $e_1 \neq e_2$.

Two expressions e_1 and e_2 with no free identifiers except model values are said to be *comparable* iff $e_1 \approx e_2$ or $e_1 \not\approx e_1$. For example 1 and "a" are not comparable, but $\{1, 2\} \not\approx \{\text{"a"}\}$ because the semantics of TLA$^+$ imply that $1 \neq 2$, and two sets are unequal if they have different cardinalities.

TLC does not handle unbounded quantifiers or unbounded CHOOSE. It assumes a particular instantiation of the bounded CHOOSE operator. That is, if we define *OneTwo* by

$$OneTwo \;\triangleq\; \textsc{choose}\ x \in \{1, 2, 3\} : x < 3$$

then TLC knows if *OneTwo* equals 1 or 2. Thus, we consider the value of the expression *OneTwo* to be comparable with 1 (as well as to every other integer).

An *elementary TLC expression* is defined to be any of the following:

- An *atomic expression*, which is either a Boolean (TRUE or FALSE), an integer contained in some finite range of integers, a string of length less than some maximum value, or a model value.
- A finite set of comparable elementary TLC expressions.
- A function $(d_1 \;{:>}\; v_1) @@ \ldots @@ (d_n \;{:>}\; v_n)$ where the d_i and v_i are elementary TLC expressions and the d_i are all unequal to one another. (The operators :> and @@ are defined in the standard *TLC* module.)

A *TLC value* is one that can be represented as an elementary TLC expression.

[2] Non-expressible values exist because there are only countably many expressible values, but ZF allows uncountable sets.

1.5 TLC Evaluation

For any constant expression e containing no free identifiers except model values, we let $(\!(e)\!)$ be the result computed by TLC when evaluating e (under the given model). This result is either an elementary TLC expression or the special symbol $\perp$, which indicates that TLC reports an error when it evaluates the expression. We say that TLC *can evaluate* an expression e iff $(\!(e)\!) \neq \perp$. Note that $(\!(\,)\!)$ maps syntactic expressions to syntactic expressions. It is idempotent: $(\!(\,(\!(e)\!)\,)\!) = (\!(e)\!)$ for any expression e, where $(\!(\perp)\!) = \perp$.

One relation between semantics and syntax that TLC is supposed to satisfy is the following soundness property:

> For any expressions e_1 and e_2 with no free identifiers other than model values, if $e_1 = e_2$, then either $(\!(e_1)\!) = (\!(e_2)\!)$ or at least one of them equals $\perp$.

1.6 Symmetry

A model can introduce model values by declaring a constant to be a model value, substituting a set of model values for a constant, and by using the Model Values section of the *Advanced Options* page of the model. The set of model values substituted for a constant can be declared to be a *symmetry set*. These symmetry declarations define a set *Sym* of permutations[3] of the model values in symmetry sets—the set consisting of all permutations P of these model values such that $P(m)$ is in the same symmetry set as m, for each model value m in a symmetry set. The set *Sym* is a group whose multiplication operation $\cdot$ is function composition—that is, $P \cdot Q$ is the permutation defined by $P \cdot Q(m) = P(Q(m))$. If the model declares no symmetry sets, then *Sym* consists only of the identity permutation.

Let the *expansion* of an expression be the expression obtained from it by recursively expanding all definitions. The expansion of an expression can contain built-in TLA$^+$ operators and constructors, model values, declared variables, bound identifiers (if the expression is a subexpression of a larger expression), and definition parameters (if it is inside the right-hand side of a definition). Since the model substitutes expressions for them, declared constants cannot appear in the expansion (though they can be instantiated by model values of the same name).

For any P in *Sym*, we extend P to arbitrary expressions by defining $P(e)$ to be the expression obtained from the expansion of e by replacing each model value m with $P(m)$. For example, if $P \in Sym$ and the m_i are model values,

[3] A permulation of a set is a 1-1 function from the set onto itself.

then:

$$\begin{aligned}&P(\{x \in \{m_1, m_2, m_3\} : x \neq m_1\})\\ &\quad = \{x \in P(\{m_1, m_2, m_3\}) : P(x \neq m_1)\}\\ &\quad = \{x \in \{P(m_1), P(m_2), P(m_3)\} : x \neq P(m_1)\}\end{aligned}$$

Note that this defines P to be a mapping from expressions to expressions. We also define $P(\perp)$ to equals $\perp$.

For P to be a useful mapping, we would expect $e_1 = e_2$ to imply $P(e_1) = P(e_2)$. However, it doesn't. For example, if there is a symmetry set with more than one element, then there exists $P \in \mathit{Sym}$ and two distinct model values m_1 and m_2 such that $P(m_1) = m_2$, $P(m_2) = m_1$, and m_1 equals $\textsc{choose}\ x \in \{m_1, m_2\} : \textsc{true}$. Then $P(m_1)$ equals m_2, but $P(\textsc{choose}\ x \in \{m_1, m_2\} : \textsc{true})$ equals $\textsc{choose}\ x \in \{P(m_1), P(m_2)\}$, which equals m_1 (because $S = T$ implies $(\textsc{choose}\ x \in S : \textsc{true}) = \textsc{choose}\ x \in T : \textsc{true}$).

This problem occurs only with CHOOSE expressions. We define an expression to be Model-Choice-Free (MCF) iff its expansion has no CHOOSE expression containing a model value. The semantics of TLA$^+$ imply:

> **Observation 1** If e_1 and e_2 are MCF, then $e_1 = e_2$ implies $P(e_1) = P(e_2)$.

TLC appears to implement evaluation so that the following is true:

> **Observation 2** TLC can evaluats an MCF expression e iff it can evaluate $P(e)$.

A defined operator is said to be MCF iff the expansion of its definition has no CHOOSE expression containing a model value or a definition parameter.

All the TLA$^+$ constructs can be viewed as syntactic sugar for built-in operators. For example, we can view $[a \mapsto x, b \mapsto y + 1]$ as an "abbreviation" for $P_{a,b}(x, y + 1)$, where $P_{a,b}$ is a built-in operator. Constructs with bound variables are abbreviations for higher-order built-in operators. For example, $\{x \in S : e\}$ is an abbreviation for $\mathit{SubsetOf}(S, \textsc{lambda}\ x : e)$ for a built-in operator $\mathit{SubsetOf}$.

An ordinary operator is one that takes only expressions (and not operators) as arguments. A higher-order operator is one that has at least one operator argument. For any ordinary defined operator Op and P in Sym, if Op is defined by

$$\mathit{Op}(a_1, \ldots, a_n) \;\triangleq\; e$$

then we define the operator $P(\mathit{Op})$ by

$$P(\mathit{Op})(a_1, \ldots, a_n) \;\triangleq\; P(e)$$

An operator Op is said to be *symmetric* iff $(\!(P(Op(e_1, \ldots, e_n)))\!)$ equals $(\!(Op(P(e_1), \ldots, P(e_n)))\!)$ for any $P \in Sym$ and MCF expressions or operators e_i. Since an expression is an operator with no arguments, this means that an expression e is symmetric iff $P(e) = e$.

All the built-in TLA$^+$ operators and constructs are symmetric except for CHOOSE. For example, the set constructor operator *SubsetOf* is symmetric because, for any MCF expression S, MCF operator Q and P in *Sym*:

$$\begin{aligned} P(\mathit{SubsetOf}(S, Q)) &= P(\{x \in S : Q(x)\}) \\ &= \{x \in P(S) : P(Q(x))\} \\ &= \{x \in P(S) : P(Q)(x)\} \quad \text{by definition of } P(Q)\text{, since } P(x) = x \\ &= \mathit{SubsetOf}(P(S), P(Q)) \end{aligned}$$

To see that CHOOSE is not symmetric, let M be a symmetry set of model values containing at least two elements and let P be a permutation of M such that $P(m) \neq m$ for all m in M. We then have:

$$\begin{aligned} &\textsc{choose}\ m \in P(M) : P(\textsc{true}) \\ &\quad = \textsc{choose}\ m \in M : \textsc{true} \qquad \text{Since } P(M) = M \text{ and } P(\textsc{true}) = \textsc{true}. \\ &\quad \neq P(\textsc{choose}\ m \in M : \textsc{true}) \quad \text{Since } P(m) \neq m \text{ for all } m \in M. \end{aligned}$$

Model values can appear in operators defined in the specification only through instantiation of constants. If all declared constants are instantiated by symmetric expressions, then any operator defined in the specification in terms of MCF expressions and MCF operators is symmetric.

1.7 Checking a Specification

1.7.1 The Properties TLC Checks

The model declares a specification *Spec*, which is a temporal formula that must be equivalent to

$$Init \land \Box[Next]_{vars} \land Fair$$

where *Init* is a state predicate, *Next* is an action, *vars* is a tuple of all declared VARIABLES, and *Fair* is usually a fairness property. (Formula *Fair* can be any formula not containing a conjunct that is a state predicate or a formula of the form $\Box[A]_v$. If it's not a fairness property, then TLC may not do what you expect it to.)

The model declares certain properties that are to be checked. We expect checking a property *Prop* to check the truth of the formula $Spec \Rightarrow Prop$. Here's what it actually does. Let the *safety specification SSpec* equal $Init \land \Box[Next]_{vars}$. TLC splits each property to be checked into the conjunction of basic properties, which it checks separately. There are four kinds of basic properties that TLC can check:

initial predicate A state predicate.

invariance A formula $\Box I$ for a state predicate I.

step simulation A formula $\Box[A]_v$ for an action A and a state expression v.

liveness A temporal formula described below that is not one of the previous three.

An invariant can be specified either in the Invariants section or the Properties part of the What to check? section of the model's *Model Overview* page. The others are specified in the Properties part.

The first three kinds of basic properties are safety properties. TLC checks such a property *Proop* by looking for a counterexample to $SSpec \Rightarrow Prop$, which is a finite (prefix of a) behavior that satisfies $SSpec \land \neg Prop$. If *Fair* is not a fairness property, then there may be a such a counterexample even if $Spec \Rightarrow Prop$ is true.

TLC checks a basic liveness property *Prop* by looking for a counterexample to $SSpec \Rightarrow (Fair \Rightarrow Prop)$. It can do this iff the formula $Fair \Rightarrow Prop$, which is equivalent to $Fair \land \neg Prop$, can be written as the conjunction of formulas, each of which is either one of the first three kinds of basic properties listed above, or else is the disjunction of conjunctions of the following kinds of formulas:

- A temporal formula containing no actions.
- A formula of the form $\Diamond\Box[A]_v$ or $\Box\Diamond[A]_v$ for an action A and a state expression v.

For example, if *Fair* equals $\mathrm{WF}_{vars}(A)$ and *Prop* equals $\mathrm{WF}_v(B)$ for actions A and B and a state expression v, then $Fair \land \neg Prop$ equals

$$(\Box\Diamond(\neg EA) \lor \Box\Diamond\langle A\rangle_{vars}) \land \neg(\Box\Diamond(\neg EB) \lor \Box\Diamond\langle B\rangle_v)$$

where EA and EB are the state predicates $\textsc{enabled}\ \langle A\rangle_{vars}$ and $\textsc{enabled}\ \langle B\rangle_v$, respectively. The tautologies $\neg\Box\Diamond F \equiv \Diamond\Box\neg F$ and $\neg\langle B\rangle_v \equiv [\neg B]_v$, together with propositional logic, imply that this formula equals

$$\begin{array}{l} \lor\ \Box\Diamond(\neg EA) \land \Diamond\Box EB \land \Diamond\Box[\neg B]_v \\ \lor\ \Box\Diamond\langle A\rangle_{vars} \land \Diamond\Box EB \land \Diamond\Box[\neg B]_v \end{array}$$

A counterexample to a property of class *liveness* is an infinite behavior.

The maximal state predicates and actions in a property are called its *atoms*. For example, $\neg EA$, EB, $\langle A\rangle_{vars}$, and $[\neg B]_v$ are the atoms of the property above.

1.7.2 The State Graph

A *TLC state* is an assignment of a TLC value to each declared variable. (Recall that a TLC value is one that can be expressed as an elementary TLC expression.) Let $\mathcal{S}$ be the set of all TLC states (for this model). (This is a countable set.) An *initial* state is an element of $\mathcal{S}$ that satisfies *Init*.

Let *StConst* and *ActConst* be the model's state and action constraints—specified by the State Constraint and Action Constraint sections of the models *Advanced Options* page, with default values TRUE.

A graph whose nodes are states in $\mathcal{S}$ is called a *state graph* for the model if it satisfies the following conditions:

1. A state s is in the graph only if s satisfies *StConst*.
2. An edge $\langle s, t \rangle$ is in the graph only if $\langle s, t \rangle$ satisfies $Next \land ActConst$ or $s = t$.[4]
3. Each state in the graph is reachable from an initial state.

The *complete* state graph is the maximal state graph, which may be finite or infinite. The nodes in the complete state graph are the *reachable* states of the model. (Because of the state and action constraints, there can be reachable states of *SSpec* that are not reachable states of the model.) A state graph is any subgraph of the complete state graph that satisfies condition 3. A behavior of the model is an infinite path in the complete state graph.

A *state tree* is a state graph that is a tree whose root is an initial state. A *state forest* is a union of disjoint (no nodes in common) trees.

Assume that each node in the complete state graph has only a finite number of outgoing edges. There are then obvious algorithms for computing the complete state graph that eventually find every reachable node and every edge joining reachable nodes. (Of course, the algorithms won't terminate if the complete state graph is infinite.) In the absence of symmetry, TLC essentially uses such an algorithm to compute approximations to the complete state graph—it uses multiple threads in an almost breadth-first search for reachable states.

1.7.3 Labels and Behaviors

Let a *state label* be an assignment of truth values to all the state-predicate atoms of the properties that TLC is checking; and let an *action label* be an assignment of truth values to all the action atoms of the properties that TLC is checking. For any state s, let $\lambda_S(s)$ be the state label that assigns to each state-predicate atom I the (truth) value of I on state s. Similarly, let $\lambda_A(s, t)$ be the action label that assigns to each action atom its value on $\langle s, t \rangle$.

[4]Note: TLC seems to add self-loops even if *ActConst* implies $var' \neq var$.

TLC can tell if a property is violated on a model behavior just knowing the labels of the behavior's nodes and edges—without knowing the actual states. TLC needs to know the states that the nodes represent only to report an error trace to the user.

The first three kinds of properties can easily be checked as the state graph is being constructed. A counterexample to a property of the fourth class is an infinite behavior. For a finite state graph, this means that it is a "lasso": a path in the state graph starting at an intial state and ending in a cycle.

Question 1.1 I believe TLC does not use constrained edges or nodes to construct a counterexample to properties of class *liveness*. Is this true?

1.7.4 Symmetry-State Graphs

For any model state s and specification variable v, let $s.v$ be the value that s assigns to v. For any $P \in \mathit{Sym}$, let $P(s)$ be the state that assigns $P(s.v)$ to every variable v. For any state s, let $[\![s]\!]$ be the set $\{P(s) : P \in \mathit{Sym}\}$. Thus, $[\![s]\!]$ is the equivalence class of s in $\mathcal{S}$ under the equivalence relation $\approx$ defined by

$$s \approx t \;\triangleq\; \exists P \in \mathit{Sym} : P(s) = t$$

(This is an equivalence relation because Sym is a group.) The set of all such equivalence classes is written $\mathcal{S}/\mathit{Sym}$.

For any e in $\mathcal{S}/\mathit{Sym}$, let $\rho(e)$ to be some state in e. That is, we assume:

$$\forall e \in \mathcal{S}/\mathit{Sym} : e = [\![\rho(e)]\!]$$

This implies that, for any s and t in $\mathcal{S}$:

$$(s \approx t) \;\equiv\; (\rho([\![s]\!]) = \rho([\![t]\!]))$$

For a state s, I will write $\rho(s)$ as an abbreviation for $\rho([\![s]\!])$. For any state s, let π_s be an element of Sym such that $\pi_s(\rho(s))$ mapping from $\mathcal{S}$ to Sym having the following property:

$$\forall s \in \mathcal{S} : s = \pi(\rho(s))$$

(The existence of $\pi(t)$ for all states t follows from the definition of ρ and the fact that Sym is a group

A *labeled symmetry graph* is a graph with

- Each node n is a record with the following components, for some state s in $\mathcal{S}$.

  class Equals $[\![s]\!]$

  Λ Equals $\lambda(s)$.

The node n is initial iff s is initial, and it is constrained iff s is constrained.

- Each edge e is a record with the following components, for some pair $\langle s, t\rangle$ of

  from An element of $\mathcal{S}/\mathit{Sym}$.

  to An element of $\mathcal{S}/\mathit{Sym}$.

  Λ A label

  Π An element of Sym.

  are 4-tuples $\langle e, f, \Lambda, \Pi$ labeled by a labeling function Λ.

whose nodes whose edges are pairs of elements in $\mathcal{S}/\mathit{Sym}$ labeled by a labeling function also called Λ and a *identifier* function Π such that, with a subset of initial nodes and constrained nodes and edges

xx

XXXXXXXXXXXXXX

We now assume that *Init*, $[\mathit{Next}]_{\mathit{vars}}$, and all atoms of the properties being checked are symmetric.

If *Init* is symmetric, then s is an initial state iff $P(s)$ is. If *Next* is symmetric, then a pair $\langle s, t\rangle$ of states satisfies *Next* iff $\langle P(s), P(t)\rangle$ does.

XXXXXXXXXXXXXXXX

For any e in $\mathcal{S}/\mathit{Sym}$, we define $\rho(e)$ to be an arbitrarily chosen state in e:

$$\rho(e) \;\triangleq\; \text{CHOOSE } s \in \mathcal{S} : e = [\![s]\!]$$

Thus, for any s and t in $\mathcal{S}$, we have

$$(s \approx t) \;\equiv\; (\rho([\![s]\!]) = \rho([\![t]\!]))$$

For a state s, I will write $\rho(s)$ as an abbreviation for $\rho([\![s]\!])$.

XXXXXXXXXXXXXX

4. Any TLA temporal formula that is the conjunction of disjunctions of the following kinds of formulas: - One containing only temporal operators and state predicates. (Those predicates can be of the form ENABLED A for an arbitrary action A.) - A formula of the form $\Box\Diamond\langle A\rangle_v$ or $\Diamond\Box[A]_{\mathit{vars}}$ for some action A and state expression v.

XXXXXXXXXXXXXXXXXXXXXXX

A *behavior* of the safety spec is a sequence $s_1, s_2, \ldots$ such that $[\![\mathit{Init}]\!](s_1)$ and $[\![[\mathit{Next}]_{\mathit{vars}}]\!](s_i, s_{i+1})$ are true for each i. Here, we consider finite behaviors as well as infinite ones. The states contained in the behaviors of the safety spec are called the *reachable* states.

$[\![\ldots]\!]$ is explained in the section on temporal logic

XXXXXXXXXXXXXXXXX

We also extend P to a permutation on the set of all states by defining $P(s)$ to be the state assigning to each variable x the value $P([\![s]\!](x))$, where $[\![s]\!](x)$ is the value that state s assigns to x.