Program Verification: Lecture 22

José Meseguer

University of Illinois at Urbana-Champaign

LTL Verification of Concurrent Programs

Modal logic can express reachability properties. But concurrent systems must also satisfy so-called liveness properties that involve infinite computations such as, e.g., (i) infinite occurrence of desired states, e.g., process non-starvation; (ii) fairness assumptions, which are crucial in many communication protocols, and (iii) infinite occurrence of desired communication patterns.

Various temporal logics extend modal logics so as to express such infinite-behavior properties. We shall study linear temporal logic (LTL), which is arguably the most user-friendly temporal logic,[1] as well as explicit-state and symbolic LTL verification methods for both declarative and imperative concurrent programs.

[1] See M. Vardi, "Branching vs. Linear Time: Final Showdown," Proc. *TACAS*, 2001, 1-22, Springer LNCS 2031, 2001.

The Syntax of $LTL(\Pi)$

Given a set Π of state predicates (also called "*atomic propositions*"), we define the formulae of the propositional linear temporal logic $LTL(\Pi)$ inductively as follows:

- **True**: $\top \in LTL(\Pi)$.
- **State predicates**: If $p \in \Pi$, then $p \in LTL(\Pi)$.
- **Next operator**: If $\varphi \in LTL(\Pi)$, then $\bigcirc\varphi \in LTL(\Pi)$.
- **Until operator**: If $\varphi, \psi \in LTL(\Pi)$, then $\varphi \, \mathcal{U} \, \psi \in LTL(\Pi)$.
- **Boolean connectives**: If $\varphi, \psi \in LTL(\Pi)$, then the formulae $\neg\varphi$, and $\varphi \vee \psi$ are in $LTL(\Pi)$.

The Syntax of $LTL(\Pi)$ (II)

Other LTL connectives can be defined as follows:

- Other Boolean connectives:
  - **False**: $\bot = \neg\top$
  - **Conjunction**: $\varphi \wedge \psi = \neg((\neg\varphi) \vee (\neg\psi))$
  - **Implication**: $\varphi \rightarrow \psi = (\neg\varphi) \vee \psi$.
- Other temporal operators:
  - **Eventually**: $\Diamond\varphi = \top\, \mathcal{U}\, \varphi$
  - **Always**: $\Box\varphi = \neg\Diamond\neg\varphi$
  - **Release**: $\varphi\, \mathcal{R}\, \psi = \neg((\neg\varphi)\, \mathcal{U}\, (\neg\psi))$
  - **Weak Until**: $\varphi\, \mathcal{W}\, \psi = (\varphi\, \mathcal{U}\, \psi) \vee (\Box\varphi)$
  - **Leads-to**: $\varphi \leadsto \psi = \Box(\varphi \rightarrow (\Diamond\psi))$
  - **Strong implication**: $\varphi \Rightarrow \psi = \Box(\varphi \rightarrow \psi)$
  - **Strong equivalence**: $\varphi \Leftrightarrow \psi = \Box(\varphi \leftrightarrow \psi)$.

The Models of LTL

The models of LTL are exactly those of modal logic, namely, Kripke structures over an alphabet Π of state predicate names. Recall from Lecture 19 that they are just triples $\mathcal{Q} = (Q, \rightarrow_{\mathcal{Q}}, \_{\mathcal{Q}})$ with $(Q, \rightarrow_{\mathcal{Q}})$ a transition system and $\_{\mathcal{Q}} : \Pi \ni p \mapsto p_{\mathcal{Q}} \in \mathcal{P}(Q)$ a meaning function interpreting each predicate name p as a subset of states $p_{\mathcal{Q}} \subseteq Q$.

The semantics of LTL is defined over maximal computation paths; that is, over sequences of state transitions that cannot be further continued. In a Kripke structure $\mathcal{Q} = (Q, \rightarrow_{\mathcal{Q}}, \_{\mathcal{Q}})$ there are two kinds of such maximal computations paths namely, (1) finite maximal paths of the form $q_0 \rightarrow_{\mathcal{Q}} q_1 \rightarrow_{\mathcal{Q}} q_2 \ldots q_{n-1} \rightarrow_{\mathcal{Q}} q_n$ with q_n a deadlock state, and (2) infinite paths of the form:

$$q_0 \rightarrow_{\mathcal{Q}} q_1 \rightarrow_{\mathcal{Q}} q_2 \ldots q_n \rightarrow_{\mathcal{Q}} q_{n+1} \ldots$$

The Models of LTL (II)

For the sake of giving a simpler LTL semantics (based only on infinite paths) we can extend any Kripke structure $\mathcal{Q} = (Q, \to_{\mathcal{Q}}, \_{\mathcal{Q}})$ to its deadlock-free extension $\mathcal{Q}^{\bullet} = (Q, \to^{\bullet}_{\mathcal{Q}}, \_{\mathcal{Q}})$, where

$$\to^{\bullet}_{\mathcal{Q}} =_{def} \to_{\mathcal{Q}} \uplus \{(q, q) \in Q^2 \mid \nexists q' \in Q \; s.t. \; q \to_{\mathcal{Q}} q'\}$$

That is, we add to $\to_{\mathcal{Q}}$ a loop transition $q \to q$ for each deadlock state q, thus making $\mathcal{Q}^{\bullet}$ deadlock free. Therefore, all maximal computation paths in $\mathcal{Q}^{\bullet}$ are infinite. By construction, the maximal paths of $\mathcal{Q}^{\bullet}$ are the infinite paths of $\mathcal{Q}$ plus the infinite paths of the form

$$q_1 \to_{\mathcal{Q}} q_2 \ldots q_{n-1} \to_{\mathcal{Q}} q_n \to q_n \to q_n \ldots$$

such that q_n is a deadlock state in $\mathcal{Q}$. In this way, both maximal finite and infinite paths of $\mathcal{Q}$ become infinite paths of $\mathcal{Q}^{\bullet}$.

Paths and Traces in a Kripke Structure

We can formalize the set of computation paths in a Kripke structure $\mathcal{Q} = (Q, \to_{\mathcal{Q}}, \_{\mathcal{Q}})$ as the set of functions:

$$Path(\mathcal{Q}) =_{def} \{\pi : \mathbb{N} \to Q \mid \forall n \in \mathbb{N},\ \pi(n) \to_{\mathcal{Q}} \pi(n+1)\}$$

Likewise, the set of computation paths in $\mathcal{Q}$ starting at state $q \in Q$ is defined as the set $Path(\mathcal{Q})_q =_{def} \{\pi \in Path(\mathcal{Q}) \mid \pi(0) = q\}$.

Given an alphabet Π of predicate symbols, the set $\mathcal{P}(\Pi)^{\omega}$ of all Π-traces is, by definition, the function set $\mathcal{P}(\Pi)^{\omega} =_{def} [\mathbb{N} \to \mathcal{P}(\Pi)]$.

Consider the function $preds : Q \ni q \mapsto \{p \in \Pi \mid q \in p_{\mathcal{Q}}\} \in \mathcal{P}(\Pi)$ maping each state q to the set of predicates holding in it. Define the set $Tr(\mathcal{Q})$ of Π-traces of $\mathcal{Q}$ by $Tr(\mathcal{Q}) =_{def} \{\pi; preds \mid \pi \in Path(\mathcal{Q})\}$. Likewise, the set $Tr(\mathcal{Q})_q$ of Π-traces starting at q is defined as $Tr(\mathcal{Q})_q =_{def} \{\pi; preds \mid \pi \in Path(\mathcal{Q})_q\}$.

The Semantics of $LTL(\Pi)$

As for modal logic, the semantics of $LTL(\Pi)$ in a Kripke structure $\mathcal{Q} = (Q, \to_{\mathcal{Q}}, \_{\mathcal{Q}})$ over predicates Π is defined by triples $\mathcal{Q}, I \models_{LTL} \varphi$, with $I \subseteq Q$ and $\varphi \in LTL(\Pi)$. By definition,

$$\mathcal{Q}, I \models_{LTL} \varphi \Leftrightarrow_{def} \forall q \in I,\ \forall \tau \in Tr(\mathcal{Q}^{\bullet})_q,\ \tau \models_{LTL} \varphi.$$

Let us unpack this definition. Is says that $\mathcal{Q}, I \models_{LTL} \varphi$ holds iff for each intial state $q \in I$ and each infinite computation path $\pi \in Path(\mathcal{Q}^{\bullet})_q$ starting at q in the deadlock-free extension $\mathcal{Q}^{\bullet}$, the trace $\tau = \pi; preds$ satisfies φ. Note, furthermore, that in the relation $\tau \models_{LTL} \varphi$ the Kripke structure $\mathcal{Q}$ has completely disappeared! Only traces are involved. The only remaining task is to define the trace satisfaction relation $\tau \models_{LTL} \varphi$ by induction on the structure of $\varphi \in LTL(\Pi)$:

- We always have $\tau \models_{LTL} \top$.

The Semantics of $LTL(\Pi)$ (II)

- For $p \in \Pi$,

$$\tau \models_{LTL} p \quad \Leftrightarrow_{def} \quad p \in \tau(0).$$

- For $\bigcirc\varphi \in LTL(\Pi)$,

$$\tau \models_{LTL} \bigcirc\varphi \quad \Leftrightarrow_{def} \quad s;\tau \models_{LTL} \varphi,$$

where $s : \mathbb{N} \longrightarrow \mathbb{N}$ is the successor function.

- For $\varphi\,\mathcal{U}\,\psi \in LTL(\Pi)$,

$$\tau \models_{LTL} \varphi\,\mathcal{U}\,\psi \quad \Leftrightarrow_{def}$$

$$(\exists n \in \mathbb{N})\ ((s^n;\tau \models_{LTL} \psi) \wedge ((\forall m \in \mathbb{N})\ m < n \ \Rightarrow s^m;\tau \models_{LTL} \varphi)).$$

- For $\neg\varphi \in LTL(\Pi)$,

$$\tau \models_{LTL} \neg\varphi \quad \Leftrightarrow_{def} \quad \tau \not\models_{LTL} \varphi.$$

The Semantics of $LTL(\Pi)$ (III)

- For $\varphi \vee \psi \in LTL(\Pi)$,

$$\tau \models_{LTL} \varphi \vee \psi \quad \Leftrightarrow_{def}$$

$$\tau \models_{LTL} \varphi \quad \text{or} \quad \tau \models_{LTL} \psi.$$

Note that, since $\mathcal{Q}^{\bullet} = (\mathcal{Q}^{\bullet})^{\bullet}$, it follows immediately from this LTL semantics that for any Kripke structure $\mathcal{Q}$ on predicates Π, set of initial states $I \subseteq Q$ and formula $\varphi \in LTL(\Pi)$ we have the equivalence:

$$\mathcal{Q}, I \models_{LTL} \varphi \;\Leftrightarrow\; \mathcal{Q}^{\bullet}, I \models_{LTL} \varphi.$$

However, the Kripke structure we have in mind is the fully general $\mathcal{Q}$, which need not be deadlock-free. $\mathcal{Q}^{\bullet}$ is just a technical device to make the definition of the $\models_{LTL}$ relation easier.

A Puzzle: $LTL(\Pi)$ is not Semantically Closed under Negation

Call a logic $\mathcal{L}$ with negation semantically closed under negation if for any model $\mathbb{M}$ and sentence φ we have the equivalence:

$$\mathbb{M} \models \neg\varphi \quad \Leftrightarrow \quad \mathbb{M} \not\models \varphi$$

where a "sentence" is a formula with no unquantified variables. Since the formulas in $LTL(\Pi)$ have no variables at all, they seem to be sentences. Yet, the above equivalence is violated. Indeed:

Consider a Kripke structure $\mathcal{Q}$ with states $Q = \{a, b, c\}$, transitions $a \to b$ and $a \to c$, $\Pi = \{p, q\}$ and with $preds(a) = preds(c) = \{p, q\}$ and $preds(b) = \{q\}$. Clearly, $\mathcal{Q}, a \not\models_{LTL} \Box p$, so we would expect to have $\mathcal{Q}, a \models_{LTL} \neg\Box p$, i.e., $\mathcal{Q}, a \models_{LTL} \Diamond\neg p$. But this is false, since it does not hold in the infinite $\mathcal{Q}^{\bullet}$ path

$$a \to c \to c \to c \ldots$$

A Puzzle: $LTL(\Pi)$ is not Semantically Closed under Negation (II)

The plot thickens if we consider the modal logic equivalence $\mathcal{Q}, a \not\models_{S4} \Box p \;\Leftrightarrow\; \mathcal{Q}, a \models_{S4} \Diamond\neg p$, plus the easy to check equivalence $\mathcal{Q}, a \not\models_{S4} \Box p \;\Leftrightarrow\; \mathcal{Q}, a \not\models_{LTL} \Box p$. They imply that $\mathcal{Q}, a \models_{S4} \Diamond\neg p \;\not\Leftrightarrow\; \mathcal{Q}, a \models_{LTL} \Diamond\neg p$, which clearly shows that there is something awry about the LTL meaning of $\Diamond\neg p$. What is it?

The puzzle's solution is that, $\mathcal{Q}, a \not\models_{LTL} \Box p$ exactly means $\exists\pi \in Path(\mathcal{Q}^\bullet)_a\; \exists n \in \mathbb{N}\; s.t.\; p \notin preds(\pi(n))$, which exactly means that $\mathcal{Q}, a \models_{S4} \Diamond\neg p$, whereas $\mathcal{Q}, a \models_{LTL} \Diamond\neg p$ exactly means that $\forall\pi \in Path(\mathcal{Q}^\bullet)_a\; \exists n \in \mathbb{N}\; s.t.\; p \notin preds(\pi(n))$.

That is, all LTL formulas are universally path quantified in an implicit manner, whereas $\Diamond\neg p$ is existensially path quantified in $\mathcal{Q}, a \models_{S4} \Diamond\neg p$. That's why $\mathcal{Q}, a \models_{S4} \Diamond\neg p \;\not\Leftrightarrow\; \mathcal{Q}, a \models_{LTL} \Diamond\neg p$.

The $LTL^+(\Pi)$ Temporal Logic

This puzzle offers an excellent opportunity, namely, to easily extend $LTL(\Pi)$ to a more expressive logic $LTL^+(\Pi)$, where both universal and existential path quantifications are allowed. Indeed, universal (**A**) and existential (**E**) path quantifiers are explicitly used in other temporal logics such as $CTL(\Pi)$ and $CTL^*(\Pi)$.[2] The definition of $LTL^+(\Pi)$ is very simple: $LTL^+(\Pi) =_{def} LTL(\Pi) \uplus \{\mathbf{E}\varphi \mid \varphi \in LTL(\Pi)\}$. This makes clear that φ abbreviates $\mathbf{A}\varphi$. $LTL^+(\Pi)$'s extended semantics just adds:

$$\mathcal{Q}, I \models_{LTL} \mathbf{E}\varphi \Leftrightarrow_{def} \exists q \in I,\ \exists \tau \in Tr(\mathcal{Q}^{\bullet})_q,\ \tau \models_{LTL} \varphi.$$

Ex.22.1. Prove that for B any Boolean combination of Π-predicates, $\mathcal{Q}, I \models_{S4} \Box B \;\Leftrightarrow\; \mathcal{Q}, I \models_{LTL} \Box B$, and $\mathcal{Q}, I \models_{S4} \Diamond B \;\Leftrightarrow\; \mathcal{Q}, I \models_{LTL^+} \mathbf{E}\Diamond B$.

[2] See, e.g., E.M. Clarke, O. Grumberg and D.A. Peled, "Model Checking," MIT Press, 2001.

Rewriting Logic as a Semantic Framework for Kripke Structures

The semantics of LTL and LTL^+ still leave open the system specification question: *How can we conveniently specify Kripke Structures*? For finite Kripke structures the answer is trivial. But the Kripke structures of most (idealized) concurrent systems are infinite, and answering well this question is a non-trivial matter.

As shown by Meseguer, Palomino and Martí-Oliet[3] any computable (in their terminology "recursive") Kripke structure has a finite specification as a computable Kripke structure $\mathbb{C}^{\Pi}_{\mathcal{R}}$ associated to an admissible rewrite theory $\mathcal{R}$. Therefore, without loss of generality we may focus on specifying Kripke structures of the form $\mathbb{C}^{\Pi}_{\mathcal{R}}$.

[3] In §4.2, Theorem 6, of "Algebraic Simulations," *J. Log. Alg. Prog.* 79, 103–143 (2010).

The Kripke Structure $\mathbb{C}_{\mathcal{R}}^{\Pi}$

Thanks to the `search` and `fvu-narrow search` commands, when verifying modal logic properties of a rewrite theory $\mathcal{R}$ there was no need to explicitly specify an alphabet Π of state predicates: any constrained constructor pattern $u \mid \varphi$ could be used as a predicate, with the predicate meaning function $\_{\mathbb{C}_{\mathcal{R}}} : (u|\varphi) \mapsto [\![u \mid \varphi]\!]$.

For LTL verification, we will also use pattern disjunctions $u_1|\varphi_1 \vee \ldots \vee u_n|\varphi_n$ as state predicates. But we need to name them by some symbol $p \in \Pi$, because such p's must appear in LTL formulas. Consequently, we will also make Π explicit in the Kripke structure $\mathbb{C}_{\mathcal{R}}^{\Pi}$. The meaning function of $\mathbb{C}_{\mathcal{R}}^{\Pi}$ will have the form:

$$\_{\mathbb{C}_{\mathcal{R}}^{\Pi}} : \Pi \ni p \mapsto (u_1|\varphi_1 \vee \ldots \vee u_n|\varphi_n) \mapsto \bigcup_{1 \leq i \leq n} [\![u_i|\varphi_i]\!] \in \mathcal{P}(C_{\Sigma/\vec{E},B,State})$$

and we will specify it equationally as explained below.

Equationally Specifying the Meaning Function $\_\mathbb{C}^{\Pi}_{\mathcal{R}}$

Suppose that $\_\mathbb{C}^{\Pi}_{\mathcal{R}}$ maps $p \in \Pi$ to $\bigcup_{1 \leq i \leq n} [\![u_i | \varphi_i]\!] \in \mathcal{P}(C_{\Sigma/\vec{E},B,State})$. This is typically an infinite set; but to use it in practice we need a finite descrition of it. How can we get it? By an admissible functional module extending the underlying equational theory $(\Sigma, E \cup B)$ of $\mathcal{R} = (\Sigma, E \cup B, R)$ into an admissible equational theory $(\Sigma, E \cup B) \subseteq (\Sigma^{\Pi}, E \cup E^{\Pi} \cup B)$ that protects $(\Sigma, E \cup B)$ and is defined as follows. W.L.O.G. we may assume that the functional module defined by $(\Sigma, E \cup B)$ itself protects BOOL. $(\Sigma^{\Pi}, E \cup E^{\Pi} \cup B)$ is obtained by adding:

- A sort *Prop* of state predicates, whose constants are the $p \in \Pi$.
- An operator $\_ \models \_ : State\ Prop \rightarrow [Bool]$ which will be used to define the meaning function $\_\mathbb{C}^{\Pi}_{\mathcal{R}}$. Note that its result sort is the kind $[Bool]$ (we assume all theories kind-complete). The reason will become clear below.

Equationally Specifying the Meaning Function $\_{\mathbb{C}^{\Pi}_{\mathcal{R}}}$ (II)

- For each $p \in P$ such that $p_{\mathbb{C}^{\Pi}_{\mathcal{R}}} = \bigcup_{1 \leq i \leq n} [\![u_i | \varphi_i]\!]$ we add the conditional equations:
  $u_1 \models p = true \;\; if \;\; \varphi_1$
  $\ldots$
  $u_n \models p = true \;\; if \;\; \varphi_n$.
  Such equations for all $p \in \Pi$ are denoted E^{Π}.

If $(\Sigma, E \cup B)$ is admissible, so is $(\Sigma^{\Pi}, E \cup E^{\Pi} \cup B)$, since: (i) the rules $\vec{E}^{\Pi}$ are sort-decreasing and terminating in one step; (ii) the (conditional) critical pairs of the rules $\vec{E}^{\Pi}$ with themselves are all joinable (all rewrite to *true*), and generate no critical pairs when compared to those in $\vec{E}$; and (iii) they are sufficiently complete by construction, since they never add junk to the sort *Bool*. This is remarkable, since $\vec{E}^{\Pi}$ only defines $u \models p$ in the positive (*true*) case.

Equationally Specifying the Meaning Function $\_{\mathbb{C}^{\Pi}_{\mathcal{R}}}$ (III)

How does $(\Sigma^{\Pi}, E \cup E^{\Pi} \cup B)$ define the meaning function $\_{\mathbb{C}^{\Pi}_{\mathcal{R}}}$? It does so because, by constuction, for each $[u] \in C_{\Sigma/\vec{E},B,State}$ and each $p \in P$ we have the equivalences:

$$[u] \in p_{\mathbb{C}^{\Pi}_{\mathcal{R}}} \quad \Leftrightarrow_{def} \quad [u] \in \bigcup_{1 \leq i \leq n} [\![u_i | \varphi_i]\!] \quad \Leftrightarrow \quad (u \models p)!_{\vec{E} \cup \vec{E}^{\Pi}/B} = true.$$

In many applications, even this very general end expressive method of defining the state predicates Π is not expressive enough. This is because, to express some useful properties, we want Π not to consists only of a finite set of constants $p_1, \ldots, p_n$, but to allow also for parametric state predicates. For example, we may need a predicate p parametric on $n \in \mathbb{N}$, i.e., to have the infinite set of predicates $\{p(n) \mid n \in \mathbb{N}\}$. We can easily extend $(\Sigma^{\Pi}, E \cup E^{\Pi} \cup B)$ for this purpose by:

Equationally Specifying the Meaning Function $\models_{\mathbb{C}^{\Pi}_{\mathcal{R}}}$ (IV)

- Adding an operator $p : s_1 \ldots s_m \to Prop$ for each predicate p parametric on data elements of sorts $s_1, \ldots, s_m$.
- Defining the meaning function for such a parametric p by equations:

  $u_1 \models p(\vec{v}_1) = true \;\; if \;\; \varphi_1$

  $\ldots$

  $u_n \models p(\vec{v}_n) = true \;\; if \;\; \varphi_n.$

  where E^{Π} now contains also such equations.

A comon case will have $p(\vec{v}_1) = \ldots = p(\vec{v}_n) = p(\vec{x})$, where $\vec{x}$ is a list of variables of sorts $s_1, \ldots, s_m$, which may also appear in the patterns $u_1, \ldots, u_n$. But the above format is more flexible. For example, we may define the meaning of the $\{p(n) \mid n \in \mathbb{N}\}$ by two equations: one for $n = 0$, and another for $n = s(k)$. Let us illustrate parametric predicates with Lecture 18's `COMM` protocol.

The COMM Protocol

```
fmod NAT-LIST is
 protecting NAT .
 sort List .
 subsorts Nat < List .
 op nil : -> List .
 op _;_ : List List -> List [assoc id: nil] .
 op |_| : List -> Nat .                              *** length function

 var N : Nat .  var L : List .
 eq | nil | = 0 .
 eq | N ; L | = s(| L |) .
 endfm

omod COMM is protecting NAT-LIST .
 protecting  QID .
 subsort Qid < Oid .
 class Sender |  buff : List, rec : Oid, cnt : Nat, ack-w : Bool .
 class Receiver |  buff : List, snd : Oid, cnt : Nat .
 msg to_from_val_cnt_ : Oid Oid Nat Nat -> Msg .
 msg to_from_ack_ : Oid Oid Nat -> Msg .
 op init : Oid Oid List -> Configuration .
```

The COMM Protocol (II)

```
vars N M : Nat . vars L Q : List .  vars A B : Oid .  var TV : Bool .

eq init(A,B,L) =  < A : Sender | buff : L, rec : B, cnt : 0, ack-w : false >
              < B : Receiver | buff : nil, snd : A, cnt : 0 > .

rl [snd] : < A : Sender | buff : (N ; L), rec : B, cnt : M, ack-w : false > =>
(to B from A val N cnt M) < A : Sender | buff : L, cnt : M, ack-w : true  > .

rl [rec] : < B : Receiver | buff : L, snd : A, cnt : M >
(to B from A val N cnt M) => (to A from B ack M)
< B : Receiver | buff : (L ; N),  snd : A, cnt : s(M) > .

rl [ack-rec] : (to A from B ack M)
               < A : Sender | buff : L, rec : B, cnt : M, ack-w : true >
=> < A : Sender | buff : L, rec : B, cnt : s(M), ack-w : false > .
endom
```

Parametric Properties and Formulas

We have a parametric family of initial states `init(A,B,L)` about which we would like to verify the following requirement:

Any initial state `init(A,B,L)` should always terminate in a state where there are no pending messages, `L` is held by `B`, `A`'s buffer is empty, and `A`'s and `B`'s counters equal the length of `L`.

Since this property is parametric on `A`, `B` and `L`, the LTL formula expressing it should also be parametric on `A`, `B` and `L`. Here is a formalization of the above requirement as a parametric formula:

$$\Diamond((\neg enabled) \wedge no.msgs \wedge holds(B, L) \wedge holds(A, nil) \wedge$$
$$(\neg waits.ack(A)) \wedge cnt(A, |L|) \wedge cnt(B, |L|)).$$

We just need to specify the formula's predicate meanings.

Specifying State Predicates in Maude

State predicates can be equationally specified by importing the following SATISFACTION module (in model-checker.maude):

```
fmod SATISFACTION is
  protecting BOOL .
  sorts State Prop .
  op _|=_ : State Prop -> Bool [frozen] .
endfm
```

We can add it to the COMM module and equationally specify all our predicates as follows:

```
in model-checker

omod COMM-PREDS is
  protecting COMM .   extending SATISFACTION .
  subsort Configuration < State .

vars N M : Nat . vars L L1 L2 Q : List .  vars A B : Oid .  var TV : Bool .
var Atts : AttributeSet .  var C : Configuration .
```

Specifying State Predicates in Maude (II)

```
*** no-messages for sender-receiver configurations and enabled predicates

ops no-msgs enabled : -> Prop [ctor] .

eq  < A : Sender | buff : L, rec : 'b, cnt : N, ack-w : TV >
 <  B : Receiver | buff : Q, snd : 'a, cnt : M > |= no-msgs = true .

eq < A : Sender | buff : (N ; L), rec : B, cnt : M, ack-w : false > C
  |= enabled = true .

eq < B : Receiver | buff : L, snd : A, cnt : M >
   (to B from A val N cnt M) C
  |= enabled = true .

eq C (to A from B ack M)
   < A : Sender | buff : L, rec : B, cnt : M, ack-w : true >
  |= enabled = true .
```

Specifying State Predicates in Maude (III)

```
*** parametric predicate: object A holds list L in its buffer

op holds : Qid List -> Prop [ctor] .

eq < A : Sender | buff : L , Atts > C |= holds(A,L) = true .
eq < B : Receiver | buff : L , Atts > C |= holds(B,L) = true .

*** parametric predicate: sender A waits for ack

op waits-ack : Qid -> Prop [ctor] .

eq < A : Sender | buff : L, rec : B, cnt : N, ack-w : TV > C
     |= waits-ack(A) = TV .

*** parametric predicate: counter's value is N in object O

op cnt : Oid Nat -> Prop [ctor] .

eq < A : Sender | cnt : N , Atts > C |= cnt(A,N) = true .
eq < B : Receiver | cnt : N , Atts > C |= cnt(B,N) = true .
endom
```

In Lecture 23 we shall model check our parametric formula.