# Program Verification: Lecture 21

José Meseguer

Computer Science Department
University of Illinois at Urbana-Champaign

## LTL Verification of Declarative Concurrent Programs

Proving that and admissible Maude system module `mod` $\mathcal{R}$ `endm` satisfies a property $\varphi$ means proving that its canonical model does:

$$\mathbb{C}_{\mathcal{R}} \models \varphi.$$

We have seen how to do this for invariants. But properties that talk about infinite behavior (e.g., fairness) require a richer logic, such as Linear Temporal Logic (LTL). Since temporal logic requires specifying state predicates that need not be specified in $\mathcal{R}$ and reasoning about infinite behaviors, we will associate to $\mathcal{R}$ a Kripke structure $\mathcal{K}(\mathcal{R}, State)_{\Pi}$ with state predicates $\Pi$ and chosen sort of states $State$. So our property satisfaction problem will be recast as:

$$\mathcal{K}(\mathcal{R}, State)_{\Pi}, [t] \models \varphi.$$

where $[t]$ is our chosen initial state of sort $State$.

## The Syntax of $LTL(AP)$

Given a set $\Pi$ 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 $LTL(AP)$ (II)

Other LTL connectives can be defined in terms of the above minimal set of connectives as follows:

- Other Boolean connectives:
  - False: $\bot = \neg\top$
  - Conjunction: $\varphi \wedge \psi = \neg((\neg\varphi) \vee (\neg\psi))$
  - Implication: $\varphi \to \psi = (\neg\varphi) \vee \psi$.

- Other temporal operators:
  - Eventually: $\Diamond\varphi = \top\,\mathcal{U}\,\varphi$
  - Henceforth: $\Box\varphi = \neg\Diamond\neg\varphi$
  - Release: $\varphi\,\mathcal{R}\,\psi = \neg((\neg\varphi)\,\mathcal{U}\,(\neg\psi))$
  - Unless: $\varphi\,\mathcal{W}\,\psi = (\varphi\,\mathcal{U}\,\psi) \vee (\Box\varphi)$
  - 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: Kripke Structures

Kripke structures are the natural models for propositional temporal logic. Essentially, a Kripke structure is a (total) unlabeled transition system to which we have added a collection of unary state predicates on its set of states.

A binary relation $R \subseteq A \times A$ on a set $A$ is called total iff for each $a \in A$ there is at least one $a' \in A$ such that $(a, a') \in R$. If $R$ is not total, it can be made total by defining $R^{\bullet} = R \cup \{(a, a) \in A^2 \mid \nexists a' \in A \; (a, a') \in R\}$. Note that a total relation $R$ is exactly a never terminating transition relation on $A$. Totality is introduced as a technical device to make all maximal (non-extensible) computations infinite.

## The Models of LTL: Kripke Structures (II)

A Kripke structure on state predicates $\Pi$ is a triple $\mathcal{A} = (A, \rightarrow_{\mathcal{A}}, \models_{\mathcal{A}})$ such that $A$ is a set, called the set of states, $\rightarrow_{\mathcal{A}}$ is a total binary relation on $A$, called the transition relation, and $\models_{\mathcal{A}}$ is a binary relation $\_ \models_{\mathcal{A}} \_ \subseteq A \times \Pi$, called the predicate satisfaction relation, specifying which state predicates $p \in \Pi$ hold in a state $a \in A$, denoted $a \models_{\mathcal{A}} p$.

How can we associate a Kripke structure to an admissible rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$? We just need to make explicit two things: (1) the intended top sort *State* of states in the signature $\Sigma$; and (2) the relevant state predicates $\Pi$. Having fixed the sort *State*, our associated Kripke structure has as its set of states the elements of sort *State* in the canonical term algebra $\mathbb{C}_{\Sigma/\vec{E},B}$.

## The Models of LTL: Kripke Structures (III)

The corresponding transition relation will the totalization $(\rightarrow_{\mathcal{R}})^{\bullet}$ of the one-step rewrite relation $\rightarrow_{\mathcal{R}}$ in the canonical model $\mathbb{C}_{\mathcal{R}} = (\mathbb{C}_{\Sigma/\vec{E},B}, \rightarrow_{\mathcal{R}})$.

We will explain later in this lecture how the remaining part of the Kripke structure —namely, the state predicates $\Pi$ and the predicate satisfaction relation $\models$ specifying what state predicates hold in each state— can also be defined using equations. However, before doing so it will be helpful to see how the semantics of LTL is defined for any Kripke structure.

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

The semantics of the temporal logic LTL is defined by means of the LTL satisfaction relation:

$$\mathcal{A}, a \models_{LTL} \varphi$$

between a Kripke structure $\mathcal{A}$ having $\Pi$ as its state predicates, a state $a \in A$, and an LTL formula $\varphi \in LTL(\Pi)$. Specifically, $\mathcal{A}, a \models_{LTL} \varphi$ holds iff for each path $\pi \in Path(\mathcal{A})_a$ the path satisfaction relation

$$\pi \models_{LTL} \varphi$$

holds, where we define the set $Path(\mathcal{A})$ of computation paths as the set of functions of the form $\pi : \mathbb{N} \longrightarrow A$ such that for each $n \in \mathbb{N}$, we have $\pi(n) \rightarrow_{\mathcal{A}} \pi(n+1)$ and define $Path(\mathcal{A})_a = \{\pi \in Path(\mathcal{A}) \mid \pi(0) = a\}$.

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

The path satisfaction relation $\pi \models_{LTL} \varphi$ is itself defined in terms of the trace satisfaction relation $\tau \models_{LTL} \varphi$, where a trace $\tau$ is a function $\tau \in [\mathbb{N} \to \mathcal{P}(\Pi)]$, i.e., a sequence:
$\tau(0),\ \tau(1),\ \tau(2),\ \ldots\ \tau(n),\ \ldots$, with each $\tau(i) \subseteq \Pi$ a subset of state predicates. $\pi \models_{LTL} \varphi$ is defined in terms of trace satisfaction by the definitional equivalence:

$$\pi \models_{LTL} \varphi \quad \Leftrightarrow_{def} \quad (\pi; \widetilde{\models}_{\mathcal{A}}) \models_{LTL} \varphi$$

where (see STACS Ex.38) $\widetilde{\models}_{\mathcal{A}} : A \ni a \mapsto \{p \in \Pi \mid a \models_{\mathcal{A}} p\} \in \mathcal{P}(\Pi)$ is the function defined by the relation $\models_{\mathcal{A}} \subseteq A \times \Pi$. So we get a trace $\mathbb{N} \xrightarrow{\pi} A \xrightarrow{\widetilde{\models}_{\mathcal{A}}} \mathcal{P}(\Pi)$. Thanks to the above equivalence, the Kripke structure $\mathcal{A}$ has dissappeared from the picture!, i.e., LTL satisfaction is defined exclusively in terms of traces $\tau \in [\mathbb{N} \to \mathcal{P}(\Pi)]$.

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

Finally, we inductively define the trace satisfaction relation for any trace $\tau \in [\mathbb{N} \to \mathcal{P}(\Pi)]$ as follows:

- We always have $\tau \models_{LTL} \top$.
- 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 $\neg\varphi \in LTL(\Pi)$,

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

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

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

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

## The LTL Module

The LTL syntax, in a typewriter approximation of the mathematical syntax, is supported in Maude by the following `LTL` functional module (in the file `model-checker.maude`).

```
mod LTL is
  protecting BOOL .
  sort Formula .

  *** primitive LTL operators
  ops True False : -> Formula [ctor format (g o)] .
  op ~_ : Formula -> Formula [ctor prec 53 format (r o d)] .
  op _/\_ : Formula Formula -> Formula [comm ctor gather (E e)
                                       prec 55 format (d r o d)] .
  op _\/_ : Formula Formula -> Formula [comm ctor gather (E e)
                                       prec 59 format (d r o d)] .
  op O_ : Formula -> Formula [ctor prec 53 format (r o d)] .
  op _U_ : Formula Formula -> Formula [ctor prec 63 format (d r o d)] .
```

```
op _R_ : Formula Formula -> Formula [ctor prec 63 format (d r o d)] .

*** defined LTL operators
op _->_ : Formula Formula -> Formula [gather (e E) prec 65
                                                   format (d r o d)] .
op _<->_ : Formula Formula -> Formula [prec 65 format (d r o d)] .
op <>_ : Formula -> Formula [prec 53 format (r o d)] .
op []_ : Formula -> Formula [prec 53 format (r d o d)] .
op _W_ : Formula Formula -> Formula [prec 63 format (d r o d)] .
op _|->_ : Formula Formula -> Formula [prec 63 format (d r o d)] .
                                                        *** leads-to
op _=>_ : Formula Formula -> Formula [gather (e E) prec 65
                                                  format (d r o d)] .
op _<=>_ : Formula Formula -> Formula [prec 65 format (d r o d)] .

vars f g : Formula .

eq f -> g = ~ f \/ g .
eq f <-> g = (f -> g) /\ (g -> f) .
eq <> f = True U f .
eq [] f = False R f .
```

```
  eq f W g = (f U g) \/ [] f .
  eq f |-> g = [](f -> (<> g)) .
  eq f => g = [] (f -> g) .
  eq f <=> g = [] (f <-> g) .

  *** negative normal form
  eq ~ True = False .
  eq ~ False = True .
  eq ~ ~ f = f .
  eq ~ (f \/ g) = ~ f /\ ~ g .
  eq ~ (f /\ g) = ~ f \/ ~ g .
  eq ~ O f = O ~ f .
  eq ~(f U g) = (~ f) R (~ g) .
  eq ~(f R g) = (~ f) U (~ g) .
endfm
```

# The `LTL` Module (II)

Note that, for the moment, no set $\Pi$ of state predicates has been specified in the `LTL` module. We will explain in what follows how state predicates are defined for a given system module `M`, and how they are added to the LTL module as a subsort `Prop` of `Formula`.

Note that the nonconstructor connectives have been defined in terms of more basic constructor connectives in the first set of equations. But since there are good reasons to put an LTL formula in negative normal form by pushing the negations next to the state predicates (this is specified by the second set of equations) we need to consider also the duals of the basic connectives $\top$, $\bigcirc$, $\mathcal{U}$, and $\vee$ as constructors. That is, we need to also have as constructors the dual connectives: $\bot$, $\mathcal{R}$, and $\wedge$ (note that $\bigcirc$ is self-dual).

## Associating Kripke Structures to Rewrite Theories

Since the models of temporal logic are Kripke structures, we need to explain how we can associate a Kripke structure to and admissible system module `mod` $\mathcal{R}$ `endm`.

We associate a Kripke structure to the rewrite theory $\mathcal{R} = (\Sigma, E, R)$ specified by such a system module by making explicit three things: (1) the intended top sort *State* of states in the signature $\Sigma$; (2) the relevant state predicates, that is, the relevant set $\Pi$ of such predicates, and (3) the satisfaction relation $\models$ between states and predicates.

In general, the state predicates need not be part of the system specification and therefore they need not be specified in our system module. They are typically part of the property specification.

## Associating Kripke Structures to Rewrite Theories (II)

This is because the state predicates need not be related to the operational semantics of a system module `M`: they are just certain predicates about the states of the system specified by `M` that are needed to specify some properties.

Therefore, after choosing a given top sort,[a] say `Foo`, in `M` as our sort *State* of states we can specify the relevant state predicates in a module `M-PREDS` which is a protecting extension of `M` according to the following general pattern:

```
mod M-PREDS is protecting M .
  including SATISFACTION .
  subsort Foo < State .
  ...
endm
```

[a] If the connected component has no top sort, we instead choose the kind `[Foo]`.

## Associating Kripke Structures to Rewrite Theories (III)

Where the dots '. . .' indicate the part in which the syntax and semantics of the relevant state predicates is specified, as further explained in what follows. The module `SATISFACTION` (which is contained in the file `model-checker.maude`) is very simple, and has the following specification:

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

where the sort `State` is unspecified. However, by importing `SATISFACTION` into `M-PREDS` and giving the subsort declaration

# Associating Kripke Structures to Rewrite Theories (IV)

```
subsort Foo < State .
```

all terms of sort `Foo` in `M` are also made terms of sort `State`. Note that we then have the kind identity, `[Foo]`$=$`[State]`.

The operator

```
op _|=_ : State Prop -> Bool [frozen] .
```

is crucial to define the semantics of the relevant state predicates in `M-PREDS`. Each such state predicate is declared as an operator of sort `Prop`.

In standard LTL propositional logic the set $\Pi$ of state predicates is assumed to be a set of constants.

## Associating Kripke Structures to Rewrite Theories (V)

In Maude we can define parametric state predicates, that is, operators of sort `Prop` which need not be constants, but may have one or more extra sorts as parameter arguments. We then define the semantics of such state predicates (when the predicate holds) by appropriate equations.

We can illustrate all this by means of a simple mutual exclusion example. Suppose that our original system module `M` is the following module `MUTEX`, in which two processes, one named `a` and another named `b`, can be either waiting or in their critical section, and take turns accessing their critical section by passing each other a different token (either `$` or `*`).

# Associating Kripke Structures to Rewrite Theories (VI)

```
mod MUTEX is
  sorts Name Mode Proc Token Conf .
  subsorts Token Proc < Conf .
  op none : -> Conf .
  op __ : Conf Conf -> Conf [assoc comm id: none] .
  ops a b : -> Name .
  ops wait critical : -> Mode .
  op [_,_] : Name Mode -> Proc .
  ops * $ : -> Token .
  rl [a-enter] : $ [a,wait] => [a,critical] .
  rl [b-enter] : * [b,wait] => [b,critical] .
  rl [a-exit] : [a,critical] => [a,wait] * .
  rl [b-exit] : [b,critical] => [b,wait] $ .
endm
```

## Associating Kripke Structures to Rewrite Theories (VII)

Our obvious sort for states is the top sort `Conf` of configurations. In order to state the desired safety and liveness properties we need state predicates telling us whether a process is waiting or is in its critical section. We can make these predicates parametric on the name of the process and define their semantics as follows:

```
mod MUTEX-PREDS is protecting MUTEX .  including SATISFACTION .
  subsort Conf < State .
  ops crit wait : Name -> Prop .
  var N : Name .
  var C : Conf .
  eq [N,critical] C |= crit(N) = true .
  eq C |= crit(N) = false [owise] .
  eq [N,wait] C |= wait(N) = true .
  eq C |= wait(N) = false [owise] .
endm
```

## Associating Kripke Structures to Rewrite Theories (VIII)

The above example illustrates a general method by which desired state predicates for a module `M` are defined in a protecting extension, say `M-PREDS`, of `M` which imports `SATISFACTION`.

One specifies the desired states by choosing a top sort in `M` and declaring it as a subsort of `State`. One then defines the syntax of the desired state predicates as operators of sort `Prop`, and defines their semantics by means of a set of equations that specify for what states a given state predicate evaluates to `true`.

We assume that those equations together with those of `M`, are ground convergent modulo $B$.

## Associating Kripke Structures to Rewrite Theories (IX)

Since we should protect `BOOL`, it is important to make sure that satisfaction of state predicates is fully defined. This can be checked with Maude's `SCC` tool.

This means that we should give equations for when the predicates are `true` and when they are `false`. In practice, however, this often reduces to specifying when a predicate is true by means of (possibly conditional) equations of the general form,

$$t \models p(v_1, \ldots, v_n) = \mathit{true} \ \ \text{if} \ \ C$$

because we can cover all the remaining cases, when it is false, with an equation

$$x : \mathit{State} \models p(y_1, \ldots, y_n) = \mathit{false} \quad [\texttt{owise}] \ \ .$$

# Associating Kripke Structures to Rewrite Theories (X)

In other cases, however —for example because we want to perform further reasoning using formal tools— we may fully define the true and false cases of a predicate not by using the `[owise]` attribute, but by explicit (possibly conditional) equations of the more general form,

$$t \models p(v_1, \ldots, v_n) = \mathit{bexp} \;\text{ if }\; C,$$

where *bexp* is an arbitrary Boolean expression.

We can now associate to an admissible system module `M` specifying a rewrite theory $\mathcal{R} = (\Sigma, E, R)$ (with a selected top sort *State* of states and with state predicates $\Pi$ defined by means of equations $D$ in a protecting extension `M-PREDS` of `M`) a Kripke structure whose set of states is $C_{\Sigma/E,\mathit{State}}$ and whose state predicates are specified by the set:

# Associating Kripke Structures to Rewrite Theories (XI)

$$\Pi_{ground} = \{\theta(p) \mid p \in \Pi,\ \theta \text{ ground substitution}\},$$

where, by convention, we use the simplified notation $\theta(p)$ to denote the ground term $p(x_1, \ldots, x_n)\theta$.

We then define the satisfaction relation $\models\ \subseteq C_{\Sigma/E,State} \times \Pi_{ground}$ by means of the definitional equivalence:

$$[u] \models \theta(p) \quad \Leftrightarrow_{def} \quad (u \models \theta(p))!_{(\vec{E}\cup\vec{D}),B} = \mathit{true}$$

where $[u] \in C_{\Sigma/E,State}$ and $\theta(p) \in \Pi_{ground}$.

The Kripke structure we are interested in is then

$$\mathcal{K}(\mathcal{R}, \mathit{State})_\Pi = (C_{\Sigma/E,State}, (\rightarrow_{\mathcal{R}})^\bullet, \models).$$