Introduction to Temporal Logic

Transition Systems, Syntax and Semantics, Specification

Ranald Clouston

April 11, 2025

Australian National University

Temporal Logic

Propositional logic has truth values that are, once chosen, fixed.

With temporal logic, propositions can change their truth values over time.

- Invented in 1950s by New Zealand logician Arthur Prior to formalise philosophical arguments.

Now a mainstay of computer science and formal computer engineering, because it can express constraints on the actions of systems over time.

c/o Wikimedia

Temporal Logics, Plural

There are many different tweaks on temporal logic we could consider, e.g.

- Base logic: intuitionistic or classical? Propositional or first order?
- Can we talk about past events, future events, or both?
- Is time considered discrete (we can talk of 'the next moment in time') or continuous?
- Can we quantify over all, or some, possible futures?

These variations all have their place, but our temporal logic will be built on a classical, propositional base, discussing the future only, with discrete time.

We look first, at a temporal logic with no quantification over possible futures, and later, at a more sophisticated one that has this feature.

- Linear Temporal Logic (LTL), then Computation Tree Logic (CTL).

A ‘Models First’ Approach

Temporal logic in industrial formal methods is usually used for model checking:

- Start with a formal description of a system (a model), and a logical statement of a desired property. Does the model have that property?
- Because models can be very complex, there is usually a focus on automated reasoning.

We will reflect this models-first emphasis by carefully defining our models before we even define the syntax of our logic.

We will, however, later talk about satisfiability (and hence validity) of temporal logic propositions, for which we do not start with a model in mind.

- We will use tableaux both to investigate satisfiability and to extract satisfying models.

References

- Michael Huth and Mark Ryan, 'Logic in Computer Science: Modelling and Reasoning about Systems', Chapter 3.
- Mark Reynolds, 'Verification via Temporal Logic: an Introduction', lecture notes.
  - I am also using his research papers 'A New Rule for LTL Tableaux' and 'Leviathan: A New LTL Satisfiability Checking Tool Based on a One-Pass Tree-Shaped Tableau', with Matteo Bertello, Nicola Gigante, and Angelo Montanari.

Transition Systems

Transition Systems

Fix a set of propositional variables *Atoms* expressing interesting facts about a system.

A transition system $\mathcal{M}$ comprises

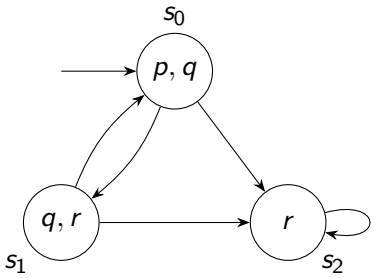
- a set S of states;
- a binary relation $\rightarrow$ on S, written infix, called the transition relation.
  - $\rightarrow$ is total, i.e. for all $s \in S$ there is at least one $s' \in S$ such that $s \rightarrow s'$.
  - note that $\rightarrow$ may be non-deterministic, i.e. there might be more than one such s'. It also might have loops $s \rightarrow s$.
- A labelling function $L : S \rightarrow \mathcal{P}(Atoms)$ specifying which propositions are true at each state.
- (often) a particular start state $s_0 \in S$.

Transition Systems, Visualised

Writing a transition system as a list of mathematical symbols is not very readable

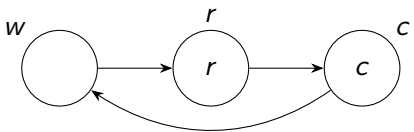
- e.g. $S = \{s_0, s_1, s_2\}$; $\rightarrow = \{(s_0, s_1), (s_0, s_2), (s_1, s_0), (s_1, s_2), (s_2, s_2)\}$; $L(s_0) = \{p, q\}, L(s_1) = \{q, r\}, L(s_2) = \{r\}$

So we instead draw a picture:



A Very Simple Meaningful Example

A machine that moves from a waiting state w, to the state of having received a request r to do some 'critical work', to doing the critical work c, then back to the waiting state.

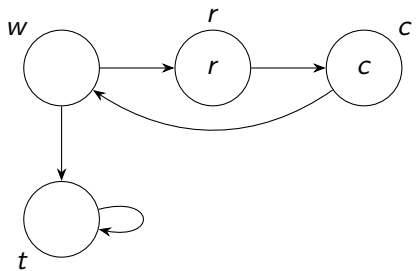


We have abused notation by reusing the names r and c both for states, and for propositions that hold only at those states.

Termination

The requirement that $\rightarrow$ be total will make some later definitions easier, but what if we want our machine to be able to terminate?

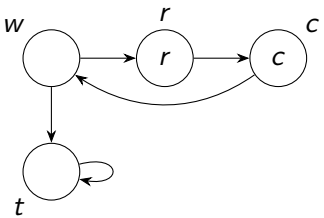
We create a new 'sink' state, with arrows into it, but no arrows out of it except a self-loop:



We can terminate, but only if we have no live request, and are not doing critical work.

Paths

A path (or fullpath) σ is an infinite sequence of related states $\sigma_0 \to \sigma_1 \to \sigma_2 \to \cdots$, often written without the $\to$ symbols. This is a possible 'run of the system', or 'possible future'.



- *wrcwrcwrcwrcwrcwrc* ...
- *wrcwrctttttttttt* ...
- *rcwttttttttttttt* ...
- *wcwcttttwwwwwwwwww* ... ✗

Paths

LTL propositions will be true or false with respect to a transition system, and a path through that system.

- e.g a proposition will be true 'always' for a given path if it holds, according to the labelling function, for all states in the path.
- Linear temporal logic cannot talk about more than one path at a time, e.g. cannot say 'there exists a path to a state satisfying proposition p'. We will need a more powerful logic for that.

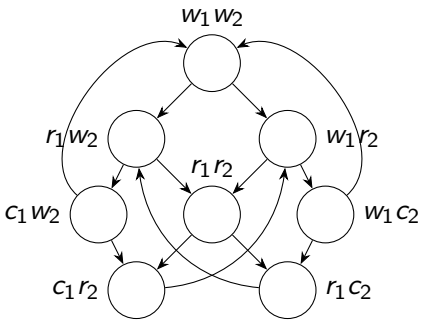
Notation: if σ is a path, and i is a natural number,

- σ_i is the i'th state of the path, e.g. σ_0 is the first state, σ_1 the second, etc.
- $\sigma_{\geq i}$ is the the path starting at σ_i and continuing as σ continued from that point.
  - e.g. $\sigma = wrcwrctttttttttt\ldots$ has $\sigma_{\geq 2} = cwrctttttttttt\ldots$, and $\sigma_{\geq 6}$ is t forever.

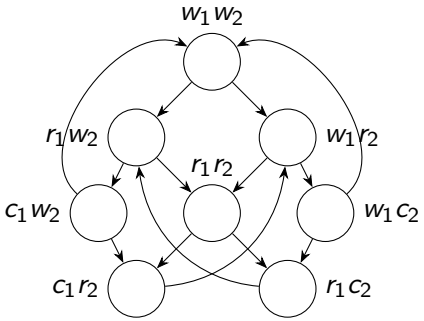
A More Complicated Meaningful Example

We met a machine that performs critical work on request; what if we had two such machines, with the constraint that they cannot perform their critical work at the same time?

- e.g. writing to a particular file, which cannot be done by two processes simultaneously.

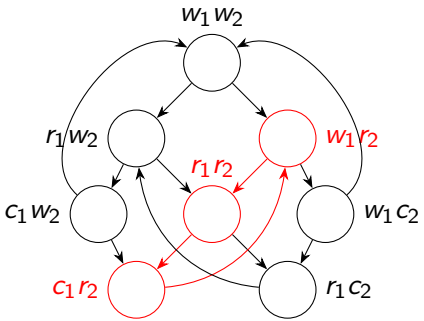


Assessing our Example



- Both machine progress correctly from waiting, to request received, to critical work.
- It clearly obeys the constraint that the two machines cannot be doing their critical work simultaneously - there is no $c_1\,c_2$ state.

Assessing our Example



- But there is a flaw: the path $w_1 r_2 \to r_1 r_2 \to c_1 r_2 \to w_1 r_2 \to \cdots$ 'starves' the second machine forever, despite its pending request.
- Avoiding this sort of bug is why we need to be able to express requirements in logic, then check them against our model.

Syntax and Semantics of Linear Temporal Logic

LTL Propositions

Definition of an LTL proposition:

$$\varphi \quad := \quad p \mid \bot \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \rightarrow \varphi \mid X\varphi \mid F\varphi \mid G\varphi \mid \varphi U \varphi$$

where p is any propositional variable.

Connective	Name	Also called
X	neXt	
F	some Future state	eventually
G	all future states	Globally, always
U	Until	

The unary X, F, G bind as tightly as $\neg$, while U binds less tightly than unary connectives but more tightly than the binary propositional connectives.

Semantics of Propositional Connectives

An LTL proposition holds, or fails to hold, for a particular transition system $\mathcal{M}$ and path σ.

We write $\models_{\mathcal{M},\sigma} \varphi$ to say that φ is satisfied by the path σ.

Semantics for the propositional connectives is defined much as for first order logic:

- $\models_{\mathcal{M},\sigma} \bot$ never
- $\models_{\mathcal{M},\sigma} \neg\varphi$ if it is not the case that $\models_{\mathcal{M},\sigma} \varphi$
- $\models_{\mathcal{M},\sigma} \varphi \wedge \psi$ if $\models_{\mathcal{M},\sigma} \varphi$ and $\models_{\mathcal{M},\sigma} \psi$
- $\models_{\mathcal{M},\sigma} \varphi \vee \psi$ if $\models_{\mathcal{M},\sigma} \varphi$ or $\models_{\mathcal{M},\sigma} \psi$ (or both)
- $\models_{\mathcal{M},\sigma} \varphi \rightarrow \psi$ if $\models_{\mathcal{M},\sigma} \varphi$ implies $\models_{\mathcal{M},\sigma} \psi$

Semantics of LTL variables

Recall that the labelling function L of the transition system tells you which variables hold at which state.

- Specifically, it maps each state to the set of variables that it satisfies.
- The labelling function says nothing about paths.

When we assess whether a propositional variable is satisfied, we do not ask anything about whether it holds in the future. We check only whether it holds right now:

- $\models_{\mathcal{M},\sigma} p$ if $p \in L(\sigma_0)$

Semantics of neXt

X talks about what will be true at the next moment:

- $\models_{\mathcal{M},\sigma} X\varphi$ if $\models_{\mathcal{M},\sigma_{\geq 1}} \varphi$

e.g.

- Xp is satisfied if the second element of the path is a state that satisfies p. Of course XXp says this for third state, etc.
- On the whiteboard, let's check the interesting fact that $\neg X\varphi$ is equivalent to $X\neg\varphi$.
- For our very simple example machine we could say $r \rightarrow X(\neg r \wedge c)$ to mean that if we start in the 'request received' state, we next proceed to the 'critical work' state.

For continuous, instead of discrete, time we would drop X from our connectives.

Semantics of 'all future states' (Globally)

- $\models_{\mathcal{M},\sigma} G\varphi$ if for all natural numbers i, $\models_{\mathcal{M},\sigma_{\geq i}} \varphi$

- All future states includes the present moment, because $\sigma_{\geq 0}$ is σ. If we wished to exclude this moment, say $XG\varphi$.
- Often we want a property to hold always, so we wrap it in a G. For example, $r \rightarrow X(\neg r \wedge c)$ only tells you something about a paths which starts in request received mode; the more general constraint is $G(r \rightarrow X(\neg r \wedge c))$.

G can be used to express safety properties: $G\neg\varphi$, where φ is some bad outcome.

- In our two machine example, if c_1, c_2 are propositions for each machine doing critical work, we would require that $G\neg(c_1 \wedge c_2)$.

Semantics of 'some Future state'

- $\models_{\mathcal{M},\sigma} F\varphi$ if there exists a natural number i such that $\models_{\mathcal{M},\sigma_{\geq i}} \varphi$

Again, this includes the present moment.

This can be used to express liveness properties: $F\varphi$, where φ is some good outcome.

- If we want φ to be 'always live', i.e. occuring infinitely often, we say $GF\varphi$.

How can we express a property that rules out the bug affecting our two machine example?

Semantics of 'some Future state'

- $\models_{\mathcal{M},\sigma} F\varphi$ if there exists a natural number i such that $\models_{\mathcal{M},\sigma_{\geq i}} \varphi$

Again, this includes the present moment.

This can be used to express liveness properties: $F\varphi$, where φ is some good outcome.

- If we want φ to be 'always live', i.e. occuring infinitely often, we say $GF\varphi$.

How can we express a property that rules out the bug affecting our two machine example?

- If machine 1 enters the 'request received' state, then it is guaranteed to eventually enter the 'critical work' state, and similarly for machine 2.

Semantics of 'some Future state'

- $\models_{\mathcal{M},\sigma} F\varphi$ if there exists a natural number i such that $\models_{\mathcal{M},\sigma_{\geq i}} \varphi$

Again, this includes the present moment.

This can be used to express liveness properties: $F\varphi$, where φ is some good outcome.

- If we want φ to be 'always live', i.e. occuring infinitely often, we say $GF\varphi$.

How can we express a property that rules out the bug affecting our two machine example?

- If machine 1 enters the 'request received' state, then it is guaranteed to eventually enter the 'critical work' state, and similarly for machine 2.
- If r_1, r_2 are the 'request received' propositions, we demand that $G(r_1 \rightarrow Fc_1)$, and that $G(r_2 \rightarrow Fc_2)$.

G vs F

G and F are interdefinable, with an assist from negation:

- $F\varphi$ is equivalent to $\neg G\neg\varphi$, and $G\varphi$ is $\neg F\neg\varphi$.
- If you apply negation to one, it turns into the other: $\neg G\varphi$ is $F\neg\varphi$, and $\neg F\varphi$ is $G\neg\varphi$.
- If you are sceptical, you can check that the semantics validates these statements for any transition system and path.

This is much like the situation with quantifiers, where $\neg\forall x\varphi$ is $\exists x\neg\varphi$ and vice versa.

- Not surprising, as the semantics of G and F are defined (in the classical metalogic) via universal and existential quantification.

In fact, our final connective will be strong enough to define both G and F.

Semantics of Until

- $\models_{\mathcal{M},\sigma} \varphi \, U \, \psi$ if there exists a natural number i such that $\models_{\mathcal{M},\sigma_{\geq i}} \psi$, and for all $h < i$ we have $\models_{\mathcal{M},\sigma_{\geq h}} \varphi$

This arguably is not a very good translation of the English word:

- 'I struggled with Logic until after the semester break' should not be translated $s \, U \, a$ (where s is struggled, and a is after the break)...
- but rather as something like $s \, U \, (a \wedge G \neg s)$?

But connectives do not have to exactly reflect English usage (a hopeless task anyway). It is enough that they have clear agreed-upon meanings in the context that they are used.

- e.g. if s is a safety condition, and g is a goal condition, then $s \, U \, g$ says we will reach our goal, and are safe until then.

From U to F and G

Write $\top$ for your favourite theorem ($\bot \to \bot$ is a nice small one). What does $\top \, U \, \varphi$ mean?

- $\vDash_{\mathcal{M},\sigma} \top \, U \, \varphi$ if...
- there exists a natural number i such that $\vDash_{\mathcal{M},\sigma_{\geq i}} \varphi$, and for all $h < i$ we have $\vDash_{\mathcal{M},\sigma_{\geq h}} \top$...
- But $\vDash_{\mathcal{M},\sigma_{\geq h}} \top$ always holds! So our original proposition holds merely if there exists a natural number i such that $\vDash_{\mathcal{M},\sigma_{\geq i}} \varphi$...
- which is exactly the condition for $\vDash_{\mathcal{M},\sigma} F\varphi$.

So we could live without $F\varphi$ and use $\top \, U \, \varphi$ instead. Similarly $G\varphi$ is $\neg F \neg \varphi$, which is $\neg(\top \, U \, \neg\varphi)$.

But we will keep F and G around because they are convenient.

Other Connectives

As with classical propositional logic, some connectives can be expressed as combinations of the others, and is a bit arbitrary what to take as basic syntax.

With propositional logic we excluded certain connectives, like iff and exclusive or, from our syntax, even though others make a different choice.

Similarly there are some temporal logic connectives you might see that we do not include:

- 'weak until' is like U except that we do not require the second proposition hold eventually; in the case that is does not, the first must hold always. So 'φ weak until ψ' can be expressed as $\varphi \, U \, \psi \vee G\varphi$.
- 'release' is the dual of U, i.e. $\neg(\neg\varphi \, U \, \neg\psi)$. This can also be understood as 'ψ weak until $(\varphi \wedge \psi)$'.

Semantics of LTL, Summarised

Leaving aside the propositional connectives,

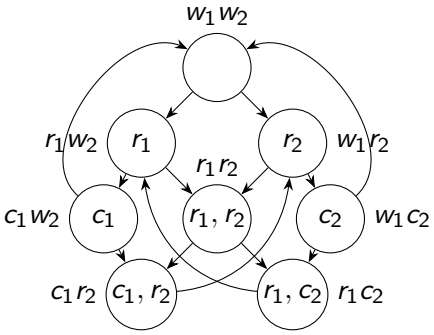
- $\vDash_{\mathcal{M},\sigma} p$ if $p \in L(\sigma_0)$
- $\vDash_{\mathcal{M},\sigma} X\varphi$ if $\vDash_{\mathcal{M},\sigma_{\geq 1}} \varphi$
- $\vDash_{\mathcal{M},\sigma} G\varphi$ if for all natural numbers i, $\vDash_{\mathcal{M},\sigma_{\geq i}} \varphi$
- $\vDash_{\mathcal{M},\sigma} F\varphi$ if there exists a natural number i such that $\vDash_{\mathcal{M},\sigma_{\geq i}} \varphi$
- $\vDash_{\mathcal{M},\sigma} \varphi\, U\, \psi$ if there exists a natural number i such that $\vDash_{\mathcal{M},\sigma_{\geq i}} \psi$, and for all $h < i$ we have $\vDash_{\mathcal{M},\sigma_{\geq h}} \varphi$

We will also write

- $\vDash_{\mathcal{M},s} \varphi$ if $\vDash_{\mathcal{M},\sigma} \varphi$ for all paths σ whose first element is s (usually, we will be interested in the start state here)
- $\vDash \varphi$ if $\vDash_{\mathcal{M},\sigma} \varphi$ for all transition systems $\mathcal{M}$ and paths σ on $\mathcal{M}$
  - e.g. $\vDash \neg X\varphi \rightarrow X\neg\varphi$ (and vice versa)

More on Specification

Returning to our Two Machine Example

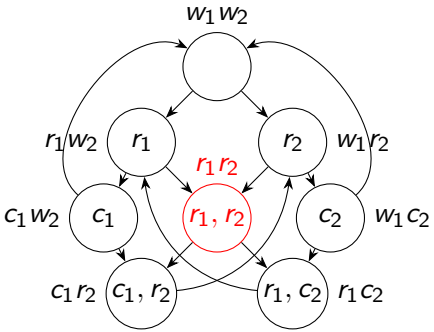


We diagnosed a problem with this transition system because it violates $G(r_1 \rightarrow Fc_1)$ and $G(r_2 \rightarrow Fc_2)$. How can we fix it?

A Fair Machine

Fairness is the condition that no process can be blocked forever by another. The easiest way to ensure this is a 'queue', so the machine that requests first, gets to act first.

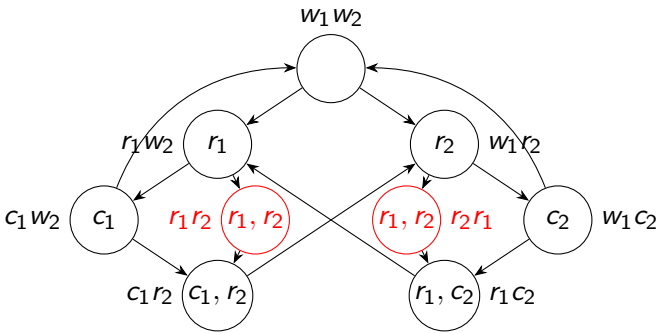
In our case we achieve this by separating the $r_1 r_2$ state into two different states, each satisfying the same propositions but recording the order in which the requests came:



A Fair Machine

Fairness is the condition that no process can be blocked forever by another. The easiest way to ensure this is a 'queue', so the machine that requests first, gets to act first.

In our case we achieve this by separating the $r_1 r_2$ state into two different states, each satisfying the same propositions but recording the order in which the requests came:



The Wolf, the Goat, and the Cabbage

An ancient puzzle - more than a thousand years old! - concerns a farmer who is trying to transport their wolf (for some reason), goat, and cabbage over the river.

c/o Illuminations

The farmer can only transport one across the river at a time, and leaving the wolf with the goat, or the goat with the cabbage, without the farmer supervising fails the task.

The Wolf, the Goat, and the Cabbage, in LTL

First choose our propositions:

- Let f, w, g, c mean the farmer, wolf, etc. is on the start side.
- Then our start state will be labelled with all four propositions, and our goal state has none of them. In logic, we want $\neg f \wedge \neg w \wedge \neg g \wedge \neg c$.

Next, start to gather our safety conditions:

- If the wolf and goat are together on the start side, then our farmer must be also: $w \wedge g \rightarrow f$. Similarly for our goal side, $\neg w \wedge \neg g \rightarrow \neg f$, and for the goat and cabbage.
- The farmer must be on every crossing: $w \wedge X\neg w \rightarrow f \wedge X\neg f$, and similarly for a goal-to-start crossing, and for the goat and cabbage.
- If the wolf changes sides, then the goat and cabbage do not move: $w \wedge X\neg w \rightarrow ((g \wedge Xg) \vee (\neg g \wedge X\neg g)) \wedge ((c \wedge Xc) \vee (\neg c \wedge X\neg c))$, and so on.

The Wolf, the Goat, and the Cabbage, in LTL

We conjoin our safety conditions with many uses of $\wedge$; call this big conjunction s.

Then we state the full list of requirements for our system as

$$f \wedge w \wedge g \wedge c \ \wedge \ s \, U \, (\neg f \wedge \neg w \wedge \neg g \wedge \neg c)$$

We can assess a putative solution against our LTL formula: