LTL patterns

- p will be false after q:

$$G(q \rightarrow G\neg p)$$

- p becomes true before q:

$$\neg q \; W \; (p \wedge \neg q)$$

$$\phi \; W \; \psi \stackrel{\text{def}}{=} (G \, \phi) \vee (\phi \; U \; \psi)$$

- p is true between q and r:

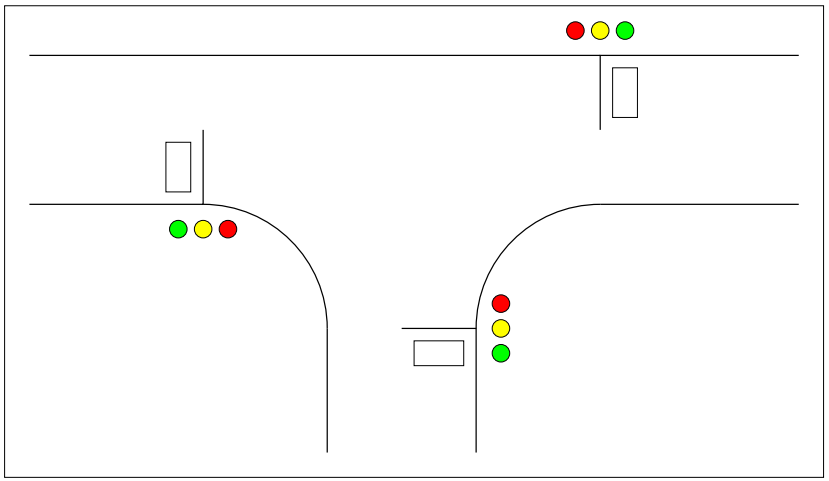
$$G((q \wedge \neg r \wedge F \, r) \rightarrow (p \; U \; r))$$

- See `http://patterns.projects.cis.ksu.edu/` for more.

Use cases

- Most specifications also provide examples.
- In CTL you can express a property that says that a use case is possible.
- In LTL that property cannot be expressed.
  However, you can express in LTL the impossibility of a use case.
  Failure, of such an absense property will tell you that the use case is possible.

Requirements for traffic lights

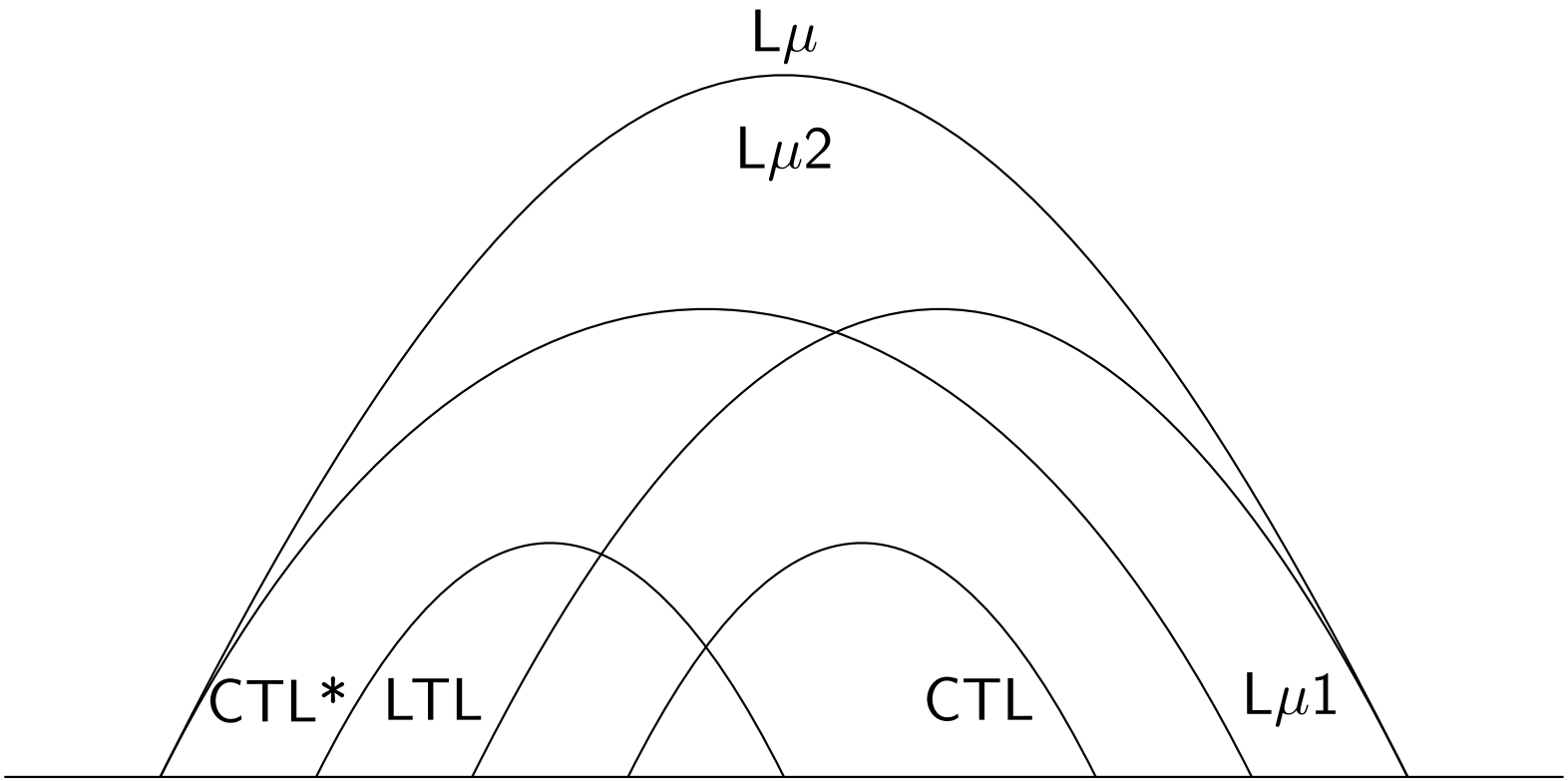


- Variables ($i \in \{1, 2, 3\}$ represent the direction):

boolean	R_i, Y_i, G_i	states of the colored lights
boolean	S_i	states of the car-waiting sensors
timer	t_i	set to 0 when a car arrives

- What are the requirements?

Expressiveness

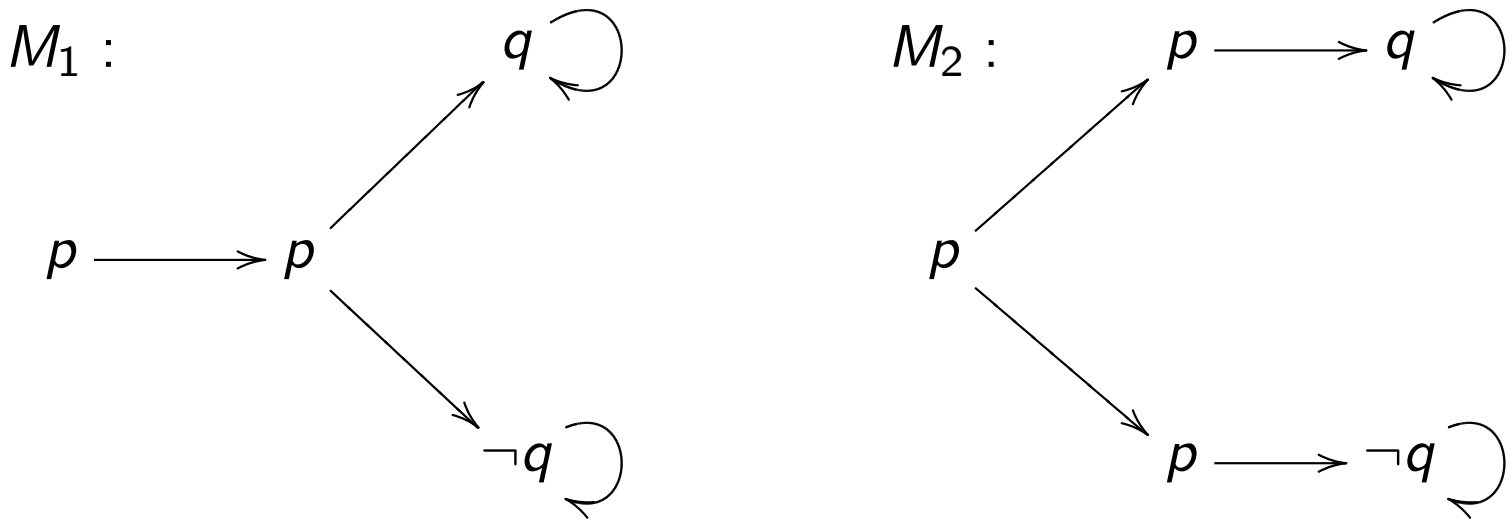


- Lμ1 Alternation free μ-calculus
- Lμ2 μ-calculus with one alternation
- Lμ μ-calculus

Limitations of LTL - Example 1

- Every LTL formula, which holds in M_1 holds in M_2.
  (The possible computations of M_1 are $\neg p^\omega$ and $\neg p^* \, p^\omega$.
  The only computation of M_2 is $\neg p^\omega$.)
- The CTL formula $\mathsf{AG}\,\mathsf{EF}\,p$ holds in M_1, but not in M_2.

Example 2



- Every LTL formula, which holds in M_1 iff it holds in M_2.
  (The computations of both M_1 and M_2 are $p\,p\,q^\omega, p\,p\,\neg q^\omega$)
- The CTL formula $\mathrm{AG}(p \rightarrow \mathrm{EF}\,q)$ holds in M_1, but not in M_2.

Leader Election

- Problem of selecting a unique leader among several peers.
- Unique identifiers for peers helps.
- Network topology can complicate matters.
- Potential for clients to crash complicates matters.

IEEE 1394 'firewire'

- Is a network with a tree topology.
- Nodes do not have a unique identifier.

IEEE 1394 leader election

The basic idea is to ask another node to be leader.

- First, a node waits for all but one of it's neighbours to ask it to be leader.
  While waiting these requests are acknowledged.
- Second, the node sends a request to the remaining neighbour.
  Unless, a request from the other node arrives first. In that case the request must be acknowledged and the node becomes the leader.
  If an acknowledgement arrives then some other node will be leader.
  If a request arrives after the request has been sent, it must be ignored.
- In this case the node waits for a random timeout (either short or long).
  If a request arrives during waiting, the node acknowledges and becomes the leader.
  Otherwise, it tries the second step again.

How to verify using model checking?

- Cannot do it for all networks: there are infinitely many.
  - Select a few networks, randomly or by hand.
  - Verify all networks up to a certain size.
- What are the properties we need to verify?
  - Can we express them all in Uppaal query language?
  - Are there variables that we need to add, just for checking?