# COL750: Foundations of Automatic Verification (Jan-May 2023)

Lectures 03 & 04 (LTL and NuSMV)

Kumar Madhukar

madhukar@cse.iitd.ac.in

Jan 12th and 16th

# Linear-time Temporal Logic

- has connectives that allow us to refer to the future
- models time as a sequence of states, extending infinitely into the future
- sequence of states is called a computational path
- since the future is not determined, we consider all possible paths

# Syntax

See Sect. 3.2.1 of the Logic in Computer Science book by Huth and Ryan.

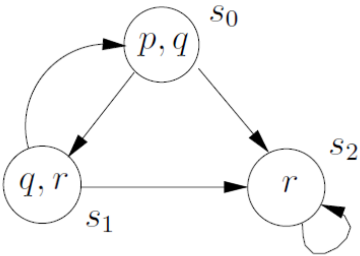
# Syntax

- Well-formed formulas
- Binding priorities
- Parse trees
- Subformulas of an LTL formula

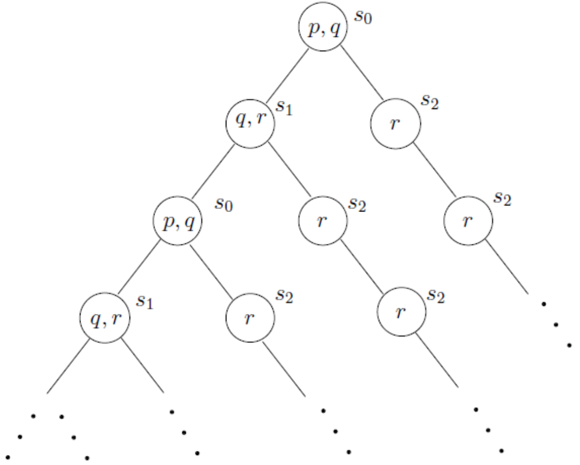
# Syntax

- Well-formed formulas
- Binding priorities
- Parse trees
- Subformulas of an LTL formula
- Subformulas of `p U (q U r)`

# Semantics



# Semantics



# Semantics

Let $\mathcal{M} = (\mathcal{S}, \rightarrow, \mathcal{L})$ be a model and $\pi = s_1 \rightarrow s_2 \rightarrow \dots$ be a path in $\mathcal{M}$.

Whether $\pi$ satisfies an LTL formula is defined by the satisfaction relation $\vDash$ as follows:

$$
\begin{aligned}
&\pi \vDash \top \\
&\pi \nvDash \bot \\
&\pi \vDash p \text{ iff } p \in L(s_1) \\
&\pi \vDash \neg\phi \text{ iff } \pi \nvDash \phi \\
&\pi \vDash \phi_1 \wedge \phi_2 \text{ iff } \pi \vDash \phi_1 \text{ and } \pi \vDash \phi_2 \\
&\pi \vDash \phi_1 \vee \phi_2 \text{ iff } \pi \vDash \phi_1 \text{ or } \pi \vDash \phi_2 \\
&\pi \vDash \phi_1 \rightarrow \phi_2 \text{ iff } \pi \vDash \phi_2 \text{ whenever } \pi \vDash \phi_1 \\
&\pi \vDash \mathrm{X}\,\phi \text{ iff } \pi^2 \vDash \phi \\
&\pi \vDash \mathrm{G}\,\phi \text{ iff, for all } i \geq 1,\ \pi^i \vDash \phi
\end{aligned}
$$

# Semantics

Let $\mathcal{M} = (\mathcal{S}, \rightarrow, \mathcal{L})$ be a model and $\pi = s_1 \rightarrow s_2 \rightarrow \ldots$ be a path in $\mathcal{M}$.

Whether $\pi$ satisfies an LTL formula is defined by the satisfaction relation $\vDash$ as follows:

$\pi \vDash \mathrm{F}\,\phi$ iff there is some $i \geq 1$ such that $\pi^i \vDash \phi$
$\pi \vDash \phi\,\mathrm{U}\,\psi$ iff there is some $i \geq 1$ such that $\pi^i \vDash \psi$ and for all $j = 1, \ldots, i-1$ we have $\pi^j \vDash \phi$

# Semantics

Let $\mathcal{M} = (\mathcal{S}, \rightarrow, \mathcal{L})$ be a model and $\pi = s_1 \rightarrow s_2 \rightarrow \ldots$ be a path in $\mathcal{M}$.

Whether $\pi$ satisfies an LTL formula is defined by the satisfaction relation $\vDash$ as follows:

$\pi \vDash \phi \text{ W } \psi$ iff either there is some $i \geq 1$ such that $\pi^i \vDash \psi$ and for all $j = 1, \ldots, i-1$ we have $\pi^j \vDash \phi$; or for all $k \geq 1$ we have $\pi^k \vDash \phi$
$\pi \vDash \phi \text{ R } \psi$ iff either there is some $i \geq 1$ such that $\pi^i \vDash \phi$ and for all $j = 1, \ldots, i$ we have $\pi^j \vDash \psi$, or for all $k \geq 1$ we have $\pi^k \vDash \psi$.

# Example specifications

- it is impossible to get to a state where `started` holds, but `ready` does not hold

# Example specifications

- for any state, if a `request` occurs, then it will eventually be granted

# Example specifications

- a certain process is `enabled` infinitely often on every computational path

# Example specifications

- on all paths, a certain process will eventually become `deadlocked`

# Example specifications

- if a process is enabled infinitely often, then it runs infinitely often

# Example specifications

- an upward travelling lift at the second floor does not change its direction when it has passengers wishing to go to the fifth floor

# Example specifications

- the lift *can* remain idle on the third floor with its doors closed

# Equivalences between LTL formulas

See Sect. 3.2.4 and Sect. 3.2.5 of the Logic in Computer Science book by Huth and Ryan.

# Verification using LTL – Example (Mutual Exclusion)

- when concurrent processes share a resource, it may be necessary to ensure that they do not have access to it at the same time
- identify certain critical sections of each process' code
- ensure that only one process is in its critical section at a time

# Verification using LTL – Example (Mutual Exclusion)

- when concurrent processes share a resource, it may be necessary to ensure that they do not have access to it at the same time
- identify certain critical sections of each process' code
- ensure that only one process is in its critical section at a time
- how do we ensure this?
- protocol for determining which process is allowed to enter its critical section
- verify that the protocol satisfies the expected properties

# Expected properties

- Safety – only one process in its critical section at a time

# Expected properties

- Safety – only one process in its critical section at a time
- Liveness – whenever any process requests access to its critical section, it will eventually be granted the access

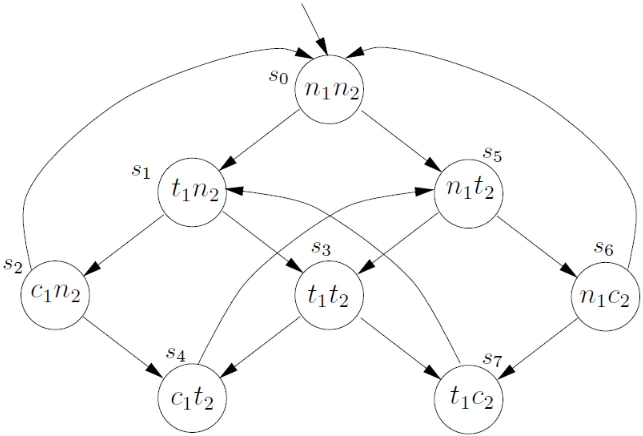
# Expected properties

- Safety – only one process in its critical section at a time
- Liveness – whenever any process requests access to its critical section, it will eventually be granted the access
- Non-blocking – a process can always request to enter its critical section

# Expected properties

- Safety – only one process in its critical section at a time
- Liveness – whenever any process requests access to its critical section, it will eventually be granted the access
- Non-blocking – a process can always request to enter its critical section
- No strict sequencing – processes need not enter their critical section in strict sequence

# Mutex protocol



# Expected properties

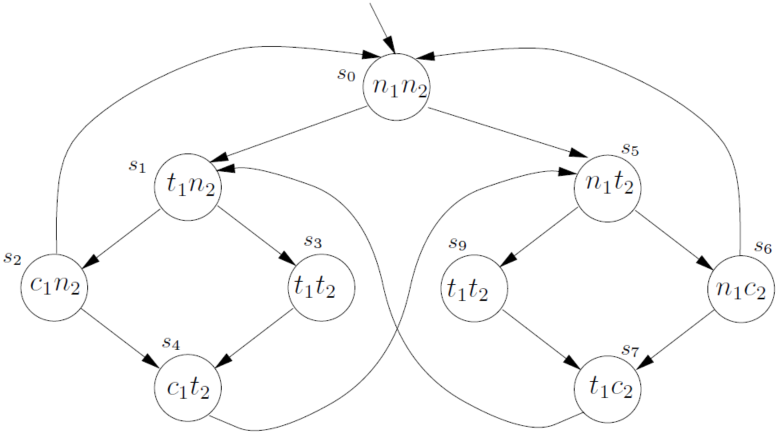
- Safety – only one process in its critical section at a time
  $\mathtt{G}\neg(c_1 \wedge c_2)$

- Liveness – whenever any process requests access to its critical section, it will eventually be granted the access
  $\mathtt{G}(t_1 \rightarrow \mathtt{F}c_1)$
  $\mathtt{G}(t_2 \rightarrow \mathtt{F}c_2)$

- Non-blocking – a process can always request to enter its critical section

- No strict sequencing – processes need not enter their critical section in strict sequence
  $\mathtt{G}(c_1 \rightarrow c_1 \ \mathtt{W} \ (\neg c_1 \wedge \neg c_1 \ \mathtt{W} \ c_2))$

# Mutex protocol revised



# NuSMV

Tool – https://nusmv.fbk.eu/

Examples done in class – https://kumarmadhukar.github.io/courses/verification-holi23/resources/nusmv-examples.tar.gz

Thank you!