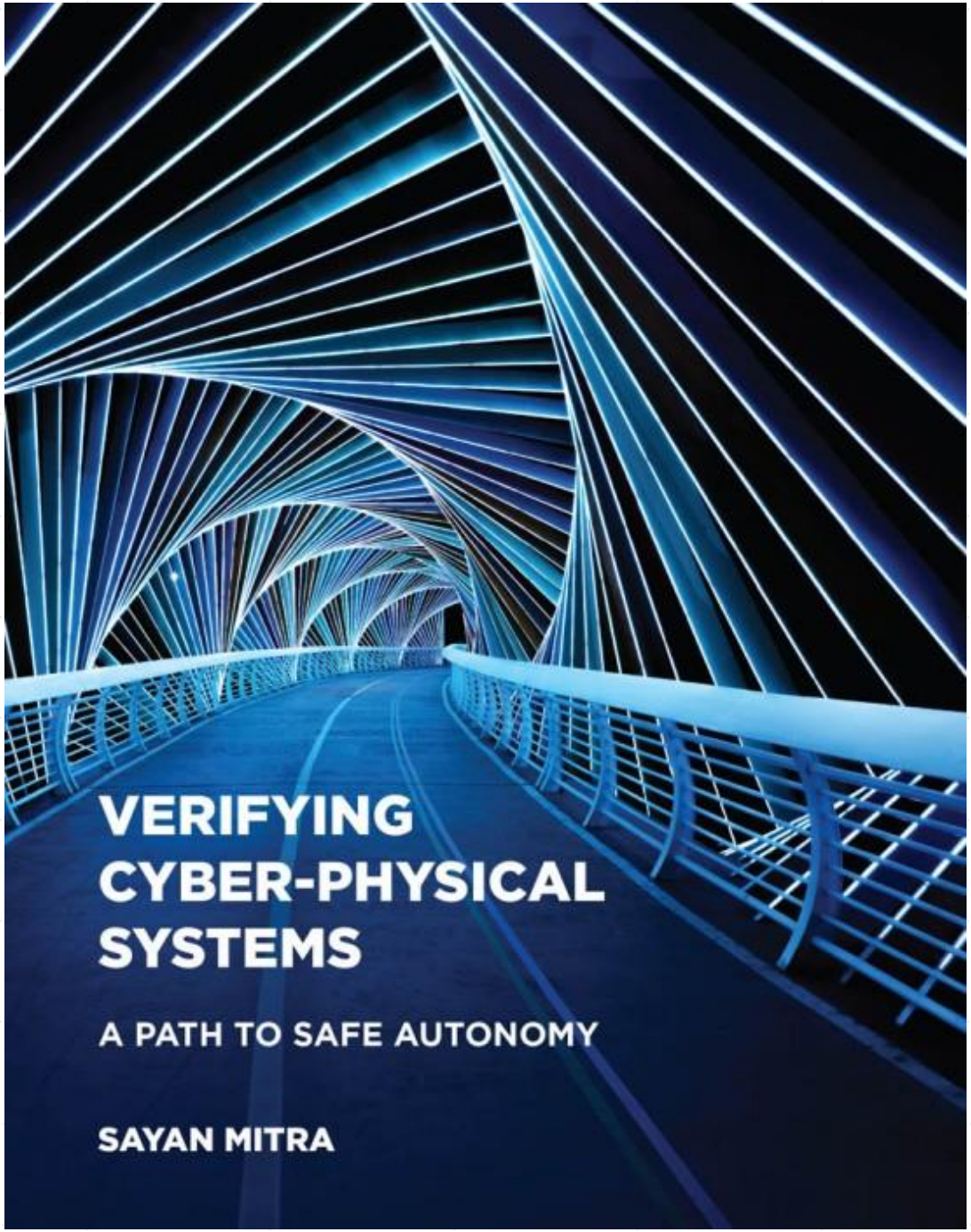
Chapter 6:
Specifying Requirements

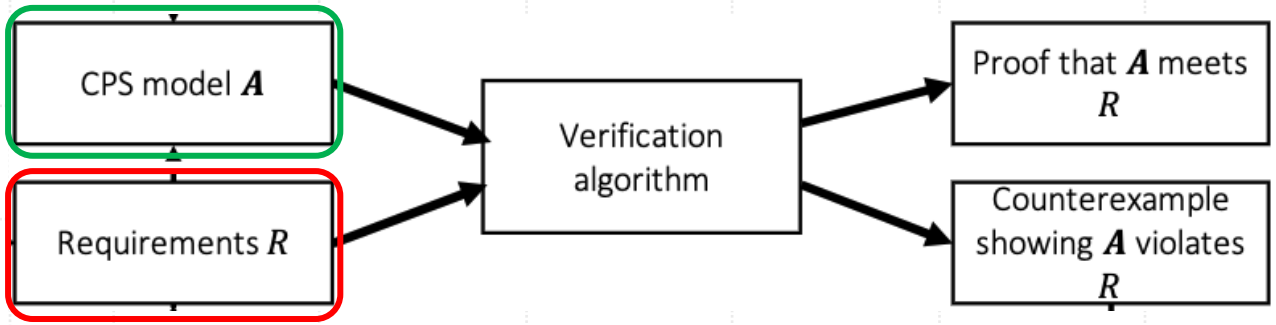
Abenezer Taye

Apr 20, 2022

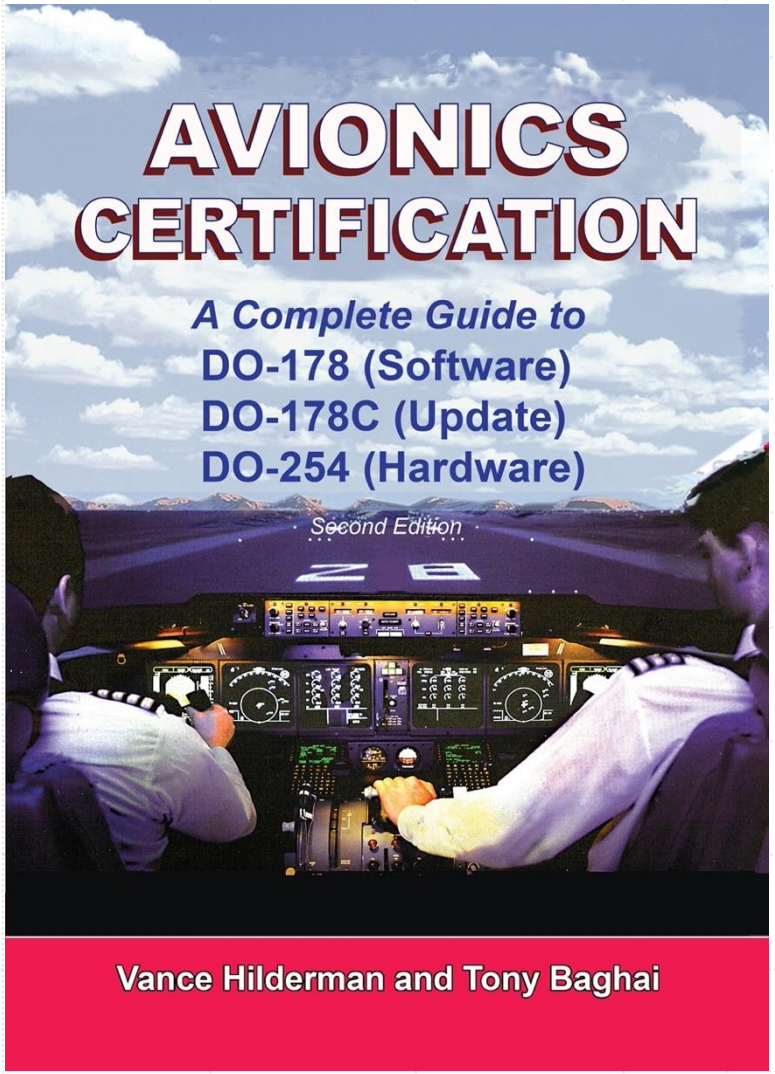


Safety Standards

- The verification process looks like:



- **Design requirement documents**: describe the **capabilities**, **functions**, and **operations** of a system.
- **Safety standards**: provide guidelines and processes for developing safety-critical systems
- **DO-178C:** a standard given by FAA for determining and certifying the airworthiness of aviation software



Safety Standards

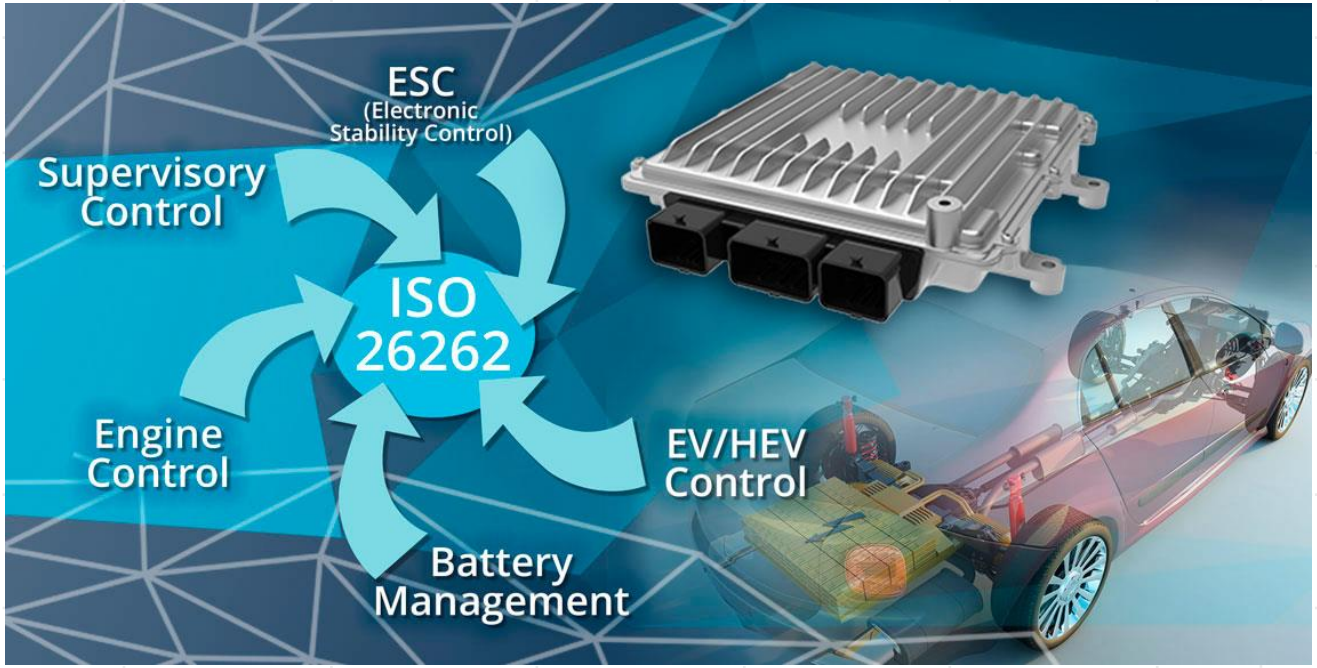
- **DO-178C:** Has five assurance levels for software modules (aka Design Assurance Levels (DALs))
- Categorizes failure conditions by their effects on the aircraft, crew, and passenger.
- Modified Condition/Decision Coverage (MC/DC) is a test suite that requires
  1. Each entry and exit point in the code be invoked
  2. Each decision take every possible value
  3. Each condition in a decision take every possible value

Design Assurance Levels (DALs) and Automotive Safety Integrity Levels (ASILs).

DAL	Failure condition	Code coverage requirements	ASIL
A	Catastrophic	MC/DC unit tests with independence, branch, functional, call, and statement coverage	N/A
B	Hazardous	Branch and statement coverage with independence, MC/DC highly recommended	ASIL D
C	Major	Statement coverage, MC/DC unit tests, branch coverage recommended	ASIL B/C
D	Minor	Statement coverage, Branch coverage recommended	ASIL A
E	No safety effect		N/A

Safety Standards

- **ISO 26262:** a standard governing the functional safety of electronic and software components in road vehicles



- Has four Automotive Safety Integrity Levels (ASILs) based on exposure to issues that affect the controllability of the vehicle

$$ASIL = (Exposure \times Controllability) \times Severity$$

- Divides probabilities of **exposure** into **four** classes very low (E1) → high (E4)

- **Four** classes to rate **controllability** → "controllable" to "difficult to control"

- **Four** classes to rate **severity** → "No injuries" to "Fatal injuries"

- A component that must be relied upon in a situation that has a **medium probability of occurrence**, and is considered **normally controllable** but can result **in life-threatening injuries** → requires an **ASIL of B**

Challenges of safety standards

- Safety standards are useful for dealing with many potential design and implementation defects.
  - However, research shows many areas of autonomous systems are not covered by existing standards
- The role of human effort in the task of verification and validation
  - Encoding the requirements and models in a verification tool
  - Validating the models (checking that the models indeed correspond to the real system)
  - Interpreting the output results of the verification tool
- There is no agreed-upon definition for precisely accounting for these human efforts

Safety requirements

- **Invariant:** captures the idea that “some things always hold” or equivalently, “bad things never happen”.
- A candidate invariant I is an invariant if all states along all executions of A satisfy I

$$\forall \alpha \in \text{Execs}_{\mathcal{A}}, \forall\, t, \alpha(t) \in I.$$

- **Progress requirements**: captures the idea that “something good eventually happens”
- Automaton A is said to meet progress requirement if every execution of A eventually reaches P.

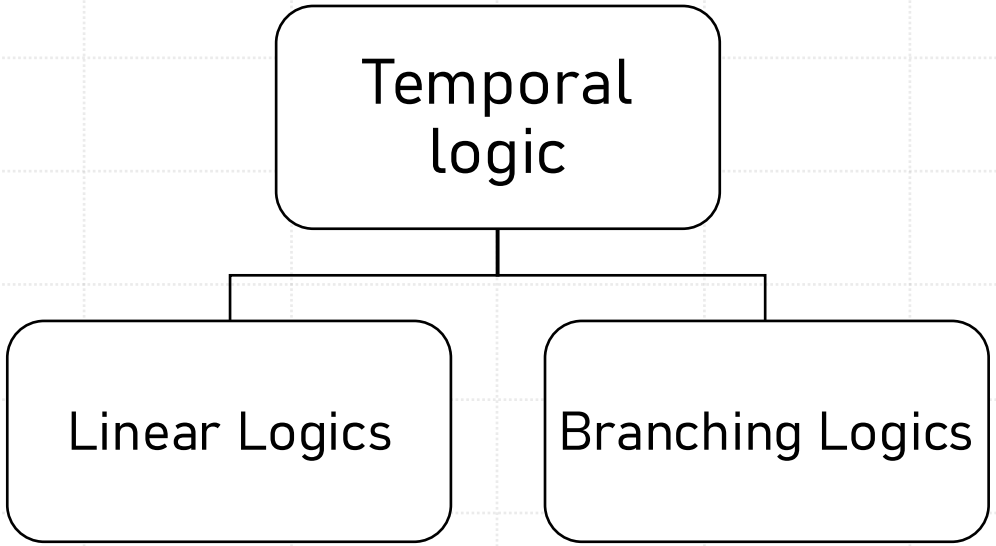
$$\forall \alpha \in \text{Execs}_{\mathcal{A}}, \exists\, t, \alpha(t) \in P.$$

- A more meaningful version of progress is captured by **asymptotic stability** → requires all executions of A to converge to P as time goes to infinity.
- A stronger version of progress bounds the time within which P must be achieved

$$\forall \alpha \in \text{Execs}_{\mathcal{A}}, \exists\, t \leq T_P, \alpha(t) \in P,$$

Formal Languages to specifying requirements

- **Temporal Logics:** a family of formal languages for succinctly specifying complex requirements
- **Temporal** refers to the sequential ordering of certain actions or predicates in the execution of an automaton



- **Branching logics**: allow quantification over execution (Computational Tree Logic (CTL))
- **Linear logics**: do not allow quantification over execution (Linear Temporal Logic (LTL))

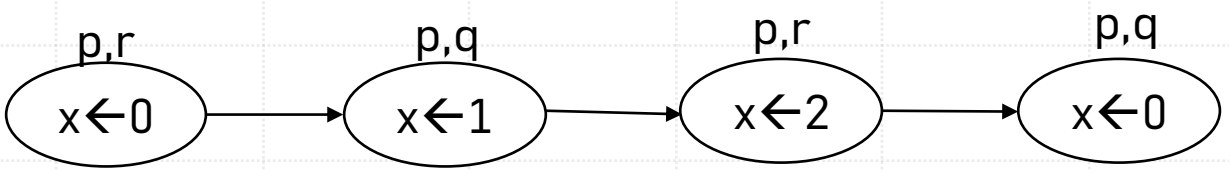
Linear Temporal Logic (LTL)

- Consider the program

```
While (x<3){
print ("hello")
if (x==1) print ("hi");
if (x==2) x=0;
else x++;
}
```

Atomic Propositions
p→ "prints hello"
q→ "prints hi"
r→ "x is even"

Temporal Patterns
- Always p holds
- Always p implies (q or r)
- Never (q and r) holds
- Always eventually q holds



- LTL semantics
  - $\neg\,(not),\ \vee\,(or),\ \wedge\,(and),\rightarrow\,(implies)$ are **propositional connectives**
  - $\circ\,(X)(next),\Diamond(F)(eventually),\ \Box\,(G)(always),(U)(until)$ are **temporal connectives**

$\neg,\ \circ,\Diamond,\ \Box$ are **unary connectives** (have high precedence than binary connectives)
$\vee,\ \wedge,\rightarrow,U$ are **binary connectives**

Linear Temporal Logic (LTL)

- $\neg\,(not),\ \vee\,(or),\ \wedge\,(and),\rightarrow(implies)$ are propositional connectives
- $\circ\,(X)(next),\Diamond(F)(eventually),\Box(G)(always),(U)$

are temporal connectives

Temporal Patterns
- Always p holds ($\Box p$)
- Always p implies (q or r) ($\Box p \rightarrow (q \vee r)$)
- Never (q and r) holds ($\neg(q \wedge r)$)
- Always eventually q holds ($\Box\Diamond q$)

Example: It is always the case that, when a lift is at the 2nd floor, travels upwards and the 5th floor is requested, it will not change direction until the 5th floor is reached.

$\Box(@2 \wedge upgoing \wedge pressed5 \rightarrow (upgoing\ U\ @5))$

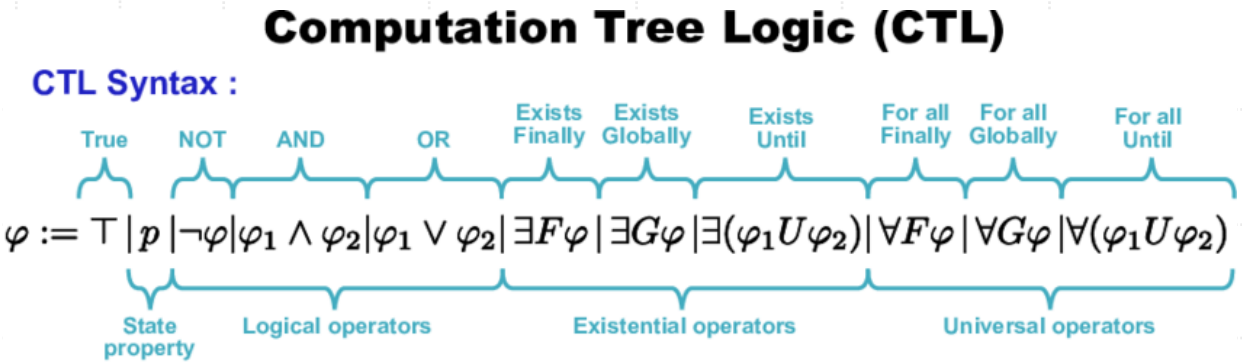
- **Key assumption in LTL:** for every state on the path, assume to know which atomic statements are true in that state
- Formal semantics

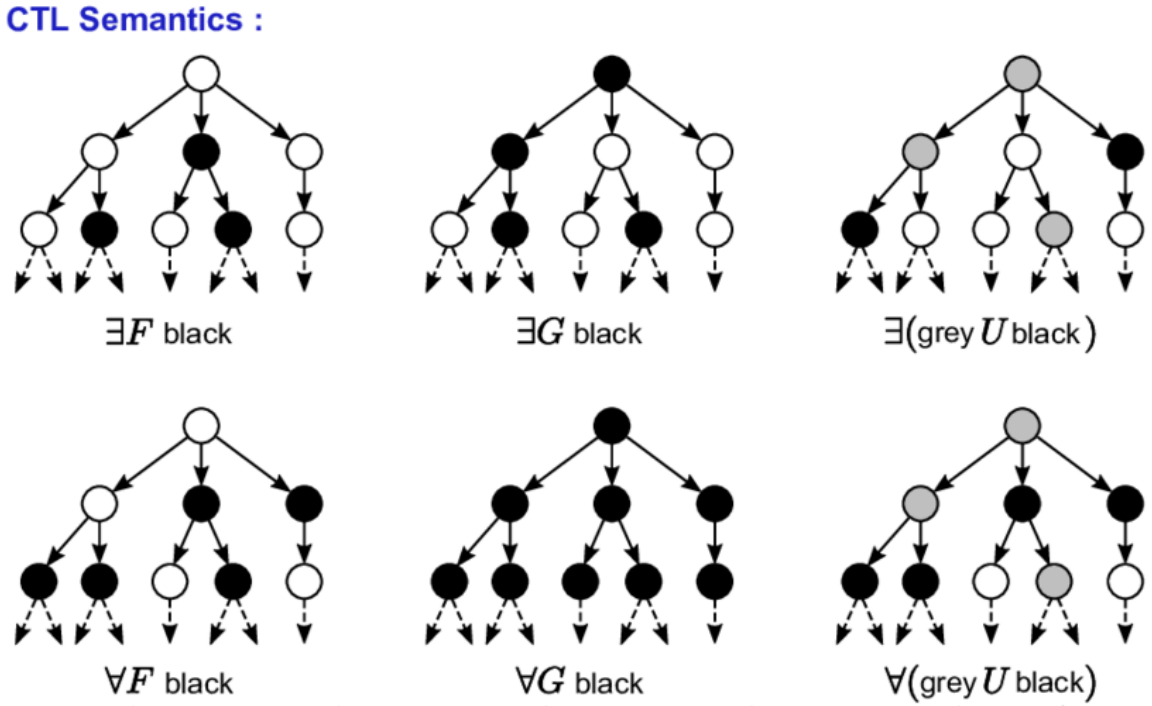
$$
\begin{aligned}
\mathbf{v} &\models true && \iff true \\
\mathbf{v} &\models p && \iff p \in Lab(q) \\
\mathbf{v} &\models \neg f && \iff \mathbf{v} \not\models f \\
\mathbf{v} &\models f_1 \wedge f_2 && \iff \mathbf{v} \models f_1 \text{ and } \mathbf{v} \models f_2 \\
\mathbf{v} &\models f_1 \vee f_2 && \iff \mathbf{v} \models f_1 \text{ or } \mathbf{v} \models f_2 \\
\mathbf{v} &\models \mathbf{X}\, f && \iff \forall \mathbf{v}' \in val(V), (\mathbf{v}, \mathbf{v}') \in \mathcal{D},\ \mathbf{v}' \models f \\
\mathbf{v} &\models \mathbf{F}\, f && \iff \forall \alpha \in \mathsf{Frags}_{\mathcal{A}}(\mathbf{v}), \exists i \geq 0, \alpha[i] \models f_2 \\
\mathbf{v} &\models \mathbf{G}\, f && \iff \forall \alpha \in \mathsf{Frags}_{\mathcal{A}}(\mathbf{v}), \forall i \geq 0, \alpha[i] \models f_2 \\
\mathbf{v} &\models f_1\ \mathbf{U}\ f_2 && \iff \forall \alpha \in \mathsf{Frags}_{\mathcal{A}}(\mathbf{v}), \exists i \geq 0, \alpha[i] \models f_2 \ \wedge\ \forall j \leq i, \alpha[j] \models f_1 \\
\mathbf{v} &\models f_1\ \mathbf{R}\ f_2 && \iff \forall \alpha \in \mathsf{Frags}_{\mathcal{A}}(\mathbf{v}), \exists i \geq 0, \alpha[i] \models f_2 \ \wedge\ \forall j \leq i, \alpha[j] \models f_1
\end{aligned}
$$

- Lab is a labeling function that assigns each state a set of atomic propositions

Computation Tree Logic (CTL)

- LTL → enables us to talk about linear or sequential requirements that must hold in every execution of a system
- CTL → allows us to describe the requirements of computational trees





CTL*: a temporal logic that freely combines temporal operators and path quantifiers → has much expressive power as both CTL and LTL

Further Discussions on Temporal Logics

- Given an LTL formula f, checking whether $[f] = \emptyset$ is called the **LTL satisfiability problem**
  - Aka LTL **model checking problem**
  - To answer this problem → we check whether there is an execution that satisfies $\neg f$
  - The problem is PSPACE-complete in the size of the formula f
- CTL and LTL have been extensively used to specify requirements for the controller synthesis problem for CPS and robotic systems
  - Problem formulation looks like this: given
    1. An LTL specification for the tasks of an agent
    2. A model of the agent's movements and actions
    3. A temporal logic specification of how the environment reacts to the agent's actions
  - Construct a controller so that the agent's resulting trajectories and actions satisfy the task and environment specifications

Further Discussions on Temporal Logics

- **Metric temporal logic (MTL):** LTL augmented with timing constraints
  - $(\Diamond_{(0,10)} lightRed)$ means lightRed will become true sometime in the next 10 time units
- **Signal temporal logic (STL):** extends MTL to include real-valued constraints
  - $\Box(x[t] < 5.5 \rightarrow (\Diamond_{(0,10)} lightRed))$
- **Probabilistic CTL (PCTL)/Continuous stochastic logics(CSL):** add probabilistic operators on top of temporal logics
  - $P_{\geq 0.5}(walkButton \rightarrow (\Diamond_{(0,10)} lightRed))$ means with a probability of at least 0.5, lightRed turns on within 10 time units of walkButton
  - Very useful while dealing a probabilistic model (like MDPs)

Thank You!!!