Systems and Software Verification

Chapter 2. Temporal Logic

Lecturer: JUNBEOM YOO
jbyoo@konkuk.ac.kr
http://dslab.konkuk.ac.kr

Ver. 2.0

2. Temporal Logic

- Motivation:
  - The elevator example includes two properties
    - "Any elevator request must ultimately be satisfied"
    - "The elevator never traverses a floor for which a request is pending without satisfying this request"
  - → Dynamic behavior of the system
  - In a first order logic,
    - $\forall t, \forall n \; (app(n, t) \Rightarrow \; \exists t' > t : serv(n, t'))$
    - $\forall t, \forall t' > t, \forall n, \quad \begin{bmatrix} (app(n, t) \wedge H(t') \neq n \wedge \; \exists t_{trav} : \\ \quad t \leq t_{trav} \leq t' \leq H(t_{trav}) = n \;) \\ \quad \Rightarrow (\exists t_{serv} : t \leq t_{serv} \leq t' \wedge serv(n, t_{serv}) \;) \end{bmatrix}$
  - But, the above notation(mathematics) is quite cumbersome.

- Temporal Logic is a different formalism, better suited for our situation.

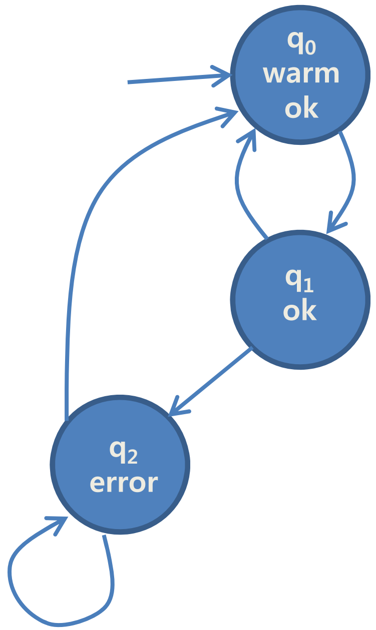
2. Temporal Logic

- Temporal Logic
  - A form of logic specifically tailored for
    - statements and reasoning
    - Involving the notion of order in time
  - Compared with the mathematical formulas
    - clearer and simpler
    - immediately ready for use (linguistic similarity of operators)
    - formal semantics (specification language tools)

- Organization of Chapter 2
  - The Language of Temporal Logic
  - The Formal Syntax of Temporal Logic
  - The Semantics of Temporal Logic
  - PLTL and CTL: Two Temporal Logics
  - The Expressivity of CTL*

2.1 The Language of Temporal Logic

- CTL*
  - serves to formally state the properties concerned with the execution of a system
  - Variants (CTL, PLTL, LTL)
  - 6 characteristics



1. Atomic Propositions

   - *warm*, *ok*, *error*

2. Proposition Formula

   - using boolean combinators
   - true, false, $\neg$, $\vee$, $\wedge$, $\Rightarrow$ (if then), $\Leftrightarrow$ (if and only if)

   - $error \Rightarrow \neg\, warm$

   (if $error$ then not $warm$)

σ_1 : (q_0: warm, ok) → (q_1: ok) → (q_0: warm, ok) → (q_1: ok) → …

σ_2 : (q_0: warm, ok) → (q_1: ok) → (q_2: error) → (q_0: warm, ok) → (q_1: ok) → …

σ_3 : (q_0: warm, ok) → (q_1: ok) → (q_2: error) → (q_2: error) → (q_2: error) → …

3. Temporal combinators

- about the sequencing of states along an execution

- X : next state
- F : a future state
- G : all the future states

- X P : the next state satisfies P
- F P : a future state satisfies P without specifying which state
  → P will hold some day (at least once)
- G P : all future states will satisfy P
  → P will always be

- $alert \Rightarrow$ F $halt$: if we are currently in a state of $alert$, then we will later be in a $halt$ state.
- G ($alert \Rightarrow$ F $halt$) : at any time, a state of $alert$ will necessarily be followed by a $halt$ state later.

- G ($warm \Rightarrow$ F $\neg warm$) : true
- G ($warm \Rightarrow$ X $\neg warm$) : true

- G is the dual of F
  - G $\phi \equiv \neg$ F $\neg\phi$

4. Arbitrary nesting of temporal combinators
    - give temporal logic its power and strength

    - GF ϕ : always there will some day be a state such that ϕ,
      ϕ is satisfied infinitely often along the execution considered
    - FG ϕ : all the time from a certain time onward, at each time instant,
      possibly excluding a finite number of instants

    - GF $warm$ ∨ FG $error$

5. U combinator
    - for until
    - ϕ_1 U ϕ_2 : ϕ_1 is verified until ϕ_2 is verified
      ϕ_2 will be verified some day, and ϕ_1 will hold in the meantime

    - G ($alert$ ⇒ ($alarm$ U $halt$)) : starting from a state of *alert*, the *alarm* remains activated until the *halt* state is eventually and inexorably reached.

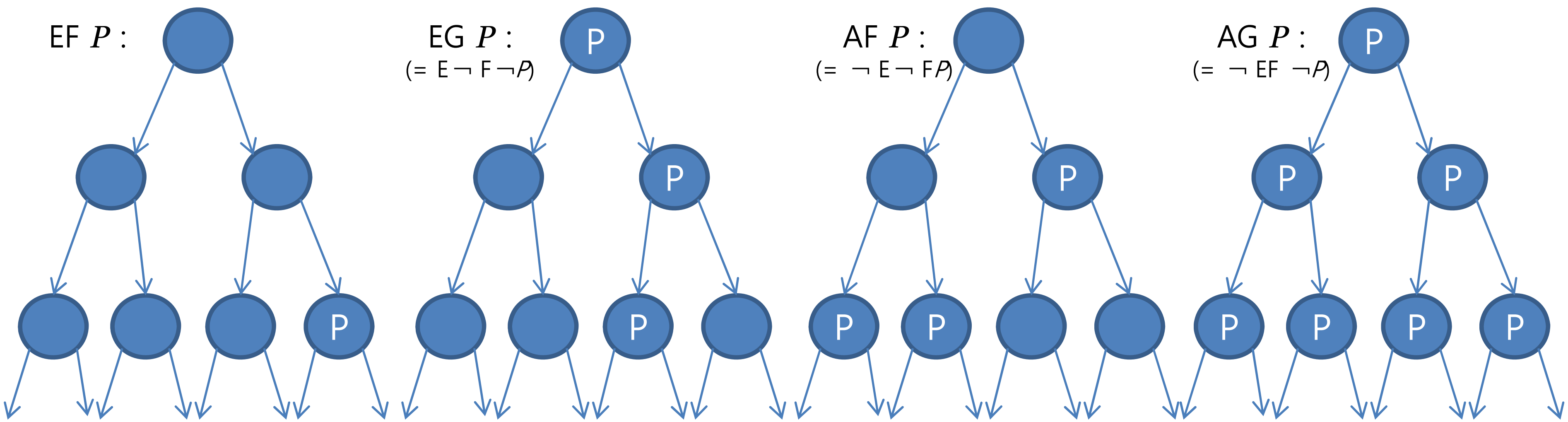
    - F ϕ ≡ $true$ U ϕ
    - ϕ_1 W ϕ_2 ≡ (ϕ_1 U ϕ_2) ∨ G ϕ_1 : weak until

6. Path quantifier

- A ϕ : all the executions out of the current state satisfy property ϕ
- E ϕ : from the current state, there exists an execution satisfying ϕ

- EF P : it is possible (by following a suitable execution) to have P some day
- EG P : there exists an execution along which P always holds

- AF P : we will necessarily have P some day (regardless of the chosen execution)
- AG P : always true



2.2 Formal Syntax of Temporal Logic

- Abstract grammar
  - Needs parentheses, operator priority, specific set of atomic propositions, etc.
  - Most model checkers use a fragment of CTL* - CTL or LTL.

$$
\begin{array}{llr}
\phi, \Psi ::= & P_1 \mid P_2 \mid \ldots & \text{(atomic proposition)} \\
& \mid \neg\phi \mid \phi \wedge \Psi \mid \phi \Rightarrow \Psi \mid \ldots & \text{(boolean combinators)} \\
& \mid X\phi \mid F\phi \mid G\phi \mid \phi \, U \, \Psi \mid \ldots & \text{(temporal combinators)} \\
& \mid E\phi \mid A\phi & \text{(path quantifiers)}
\end{array}
$$

2.3 The Semantics of Temporal Logic

- Kripke structure
  - Name of the models of temporal logic
  - Propositions labeling the states are important in CTL*
  - Transition labels (E) are neglected. $A = < Q, T, q_o , l >$, $T \subseteq Q \times Q$

- Satisfaction
  - $A,\sigma,i \models \phi$
    - "at time i of the execution σ, ϕ is true."
    - where σ is an execution of A, which not required to start at the initial state
    - A is often omitted.
  - $\sigma,i \models \phi$: ϕ is satisfied at time i of σ
  - $\sigma,i \not\models \phi$: ϕ is not satisfied at time i of σ

  - $A \models \phi$ iff $\sigma,0 \models \Phi$ for every execution of σ of A
    - "the automaton A satisfies ϕ"
    - $A \not\models \phi \quad \neq \quad A \models \neg\phi$
    - $\sigma,i \not\models \phi \quad = \quad \sigma,i \models \neg\phi$

$$
\begin{array}{ll}
\sigma, i \models P & \text{iff } P \in l(\sigma(i)), \\
\sigma, i \models \neg\phi & \text{iff it is not true that } \sigma, i \models \phi, \\
\sigma, i \models \phi \wedge \psi & \text{iff } \sigma, i \models \phi \text{ and } \sigma, i \models \psi, \\
 & \\
\sigma, i \models \mathsf{X}\phi & \text{iff } i < |\sigma| \text{ and } \sigma, i+1 \models \phi, \\
\sigma, i \models \mathsf{F}\phi & \text{iff there exists } j \text{ such that } i \leq j \leq |\sigma| \text{ and } \sigma, j \models \phi, \\
\sigma, i \models \mathsf{G}\phi & \text{iff for all } j \text{ such that } i \leq j \leq |\sigma|, \text{ we have } \sigma, j \models \phi, \\
 & \\
\sigma, i \models \phi\mathsf{U}\psi & \text{iff there exists } j,\ i \leq j \leq |\sigma| \text{ such that } \sigma, j \models \psi, \text{ and} \\
 & \quad \text{for all } k \text{ such that } i \leq k < j, \text{ we have } \sigma, k \models \phi, \\
 & \\
\sigma, i \models \mathsf{E}\phi & \text{iff there exists a } \sigma' \text{ such that } \sigma(0)\ldots\sigma(i) = \sigma'(0)\ldots\sigma'(i) \text{ and} \\
 & \quad \sigma', i \models \phi, \\
\sigma, i \models \mathsf{A}\phi & \text{iff for all } \sigma' \text{ such that } \sigma(0)\ldots\sigma(i) = \sigma'(0)\ldots\sigma'(i), \text{ we have} \\
 & \quad \sigma', i \models \phi.
\end{array}
$$

Semantics of CTL*

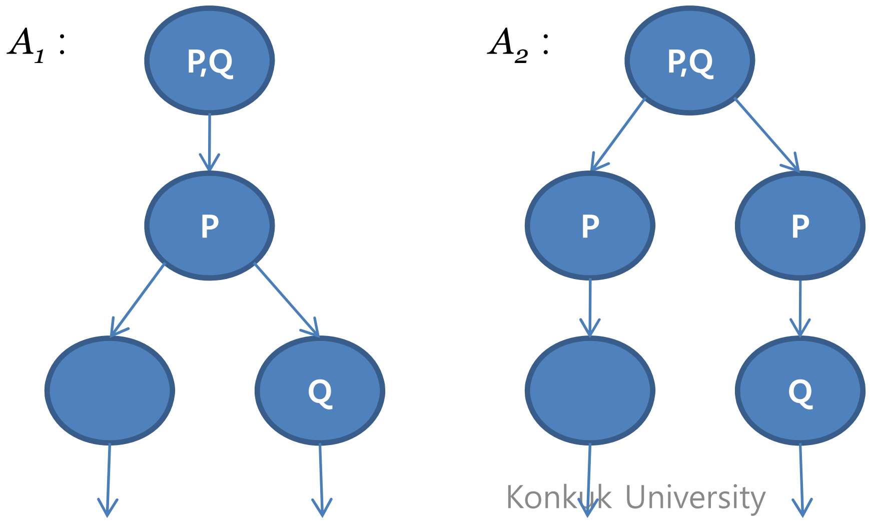
CTL*

- Time is discrete.
- Nothing exists between i and i + 1.
- The instants are the points along the executions

2.4 PLTL and CTL: Two Temporal Logics

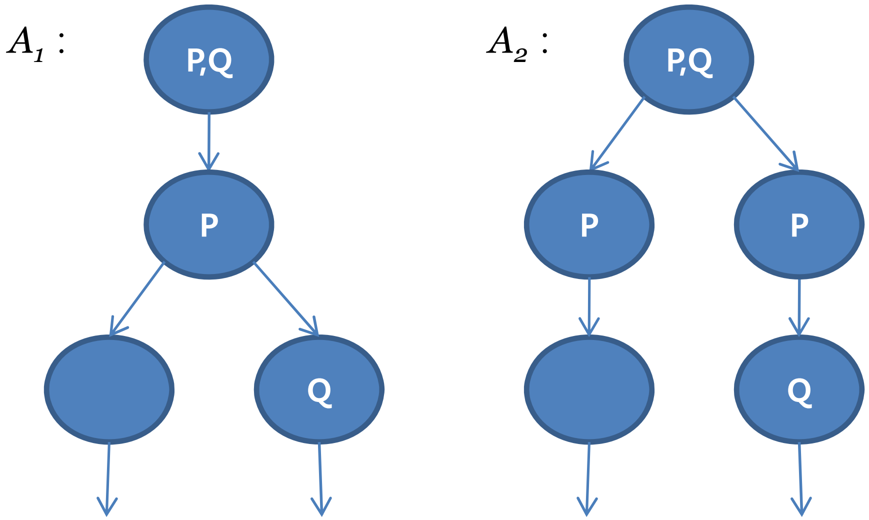
- Two most commonly used temporal logics in model checking tools
  - PLTL (Propositional Linear Temporal Logic)
  - CTL (Computational Tree Logic)
  - fragments of CTL*

- PLTL
  - No path quantifiers (A and E)
  - Linear time logic → Path formula
  - For example, PLTL cannot distinguish A_1 from A_2



Execution 1 : {P, Q} . {P}. {-}
Execution 2 : {P, Q} . {P} . {Q}

- CTL
  - Temporal combinators (X, F, U) should be under the immediate scope of path quantifier (A, E)
  - EX , AX , EU , AU , EF , EG , AG , AF , …
  - State formulas
    - Truth only depends on the current state and the automaton regions made reachable by it
    - Not depend on a current execution.
    - $q \models \phi$: ϕ is satisfied in state q
    - CTL can distinguish automata $A1$ and $A2$



$A_1, q_0 \models$ AX (EXQ $\wedge$ EX$\neg Q$)
$A_2, q'_0 \not\models$ AX (EXQ $\wedge$ EX$\neg Q$)

  - Potential reachability : AG EF P
  - Do not allow us to express very rich properties along the paths.

- Which to choose CTL or PLTL ?
  - To state some properties
    → PLTL
  - To perform exhaustive verification of a system
    → CTL
  - For both purposes
    → CTL*
    - Less popular
    - More complicated than PLTL
  - CTL + Fairness properties → FCTL
  - If we use model checking tools, then we have no choice
    - SMV : CTL (CTL*)
    - SPIN : PLTL
    - VIS : CTL / PLTL

2.5 The Expressivity of CTL*

- No logic can express anything not taken into account by the modeling decision made

- CTL* is rather expressive enough, when
  - Properties concern the execution tree of our automata

  - CTL* combinators are sufficiently expressive
  - CTL* is almost always sufficient