**Linear Temporal Logic**

**1. Basic**

> A logic for reasoning about execution paths of systems

> One of the most important logics for software and hardware verification

> Consider execution paths of a system into the future

- Lable states with atomic propositions **p, q, r, …** that hold along paths at various points in time

- LTL formulas can express **regular patterns** about these propositions as execution proceeds.

- example

while(x<3){

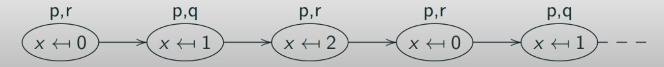
print(“Hello”); Let p be prints hello

if(x == 1) print(“hi”); q be prints hi

if(x == 2) x = 0; r be x is even.

else x++; Say we start in a state where x is 0

}



-> Always p holds.

-> Always p implies (q or r).

-> Never (q and r) holds.

-> Always eventually q holds.(좀만 기달리면 q가 나옴)

**2. Syntax**

> Assume some set Atoms of atomic propositions(atoms for short) usually denoted p, q, r etc.

> LTL formulas, usually denoted

¬, ∨, ∧ → are propositional connectives : not, or, and, implies

○, ◇, □, ∪ are temporal connectives next, eventually, always, until

- 우선순위 : (¬, ○, ◇, □) > (∨, ∧ →, ∪)

- example

: (◇p∧□q)→(p∪r) eventually p and always q implies p until r

**3. Informal Semantics**

> LTL formulas are evaluated along this path, looking into the future.

- An atomic proposition p holds if p is true at the current point in time.

- The propositional connectives ¬, ∨, ∧, → have their usual meanings, e.g., φ∧ψ holds if φ holds and ψ holds.

- Meaning of temporal connectives

: ○φ holds if φ holds **next**, i.e., at the next point in time

: △φ holds if φ holds **eventually**, i.e., now or at some future point in time

: □φ holds if φ holds **always**, i.e., now and at all future poins in time

: φ∪ψ holds if φ holds **until** ψ holds; i.e., ψ holds now or at some point in the future, and φ holds continuously until then

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

**4. Praictical Specification Patterns**

> Patterns

- A process is always active in its starting state

: □(start →active)

- It is always the case that requests are evenrually granted

: □(request → ◇grant)

- A given process will be enabled infinitely often

: □◇enabled

- If a process is enabled infinitely often, then it will run infinitely often

: □◇enabled →□◇run

- A process will never becom permanently inactive

: ¬◇□¬active

: ¬◇ = never 이랑 같음

> 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

: □(@2 ∧ pressed5 ∧upgoing → (upgoing∪@5))

**5. Formal Semantics**

> Let S be a set of states and L : S →P(Atoms) be labeling function associating to each state s a set L(s) of all atoms that are true in that state.

- P(Atoms) is the powerset(i.e., set of all subsets) of Atoms.

Let π be an infinite sequence of state s0, s1, s2, … . We think of L(si) as the set of all atoms true at point I in time on the path π.

For each I, we write πi for the i’th suffix of π, namely si, si+1, si+2, …

E.g., π1 is s1, s2, s3, …, and π2 is s2, s3, s4 …

For an LTL formula φ, we define π L φ, read “ π satisfies φ w.r.t. labeling L” of “φ holds for π w.r.t. labeling L” by structural recursion on φ :

π L p iff p∈L(s0)

π L φ∧ψ iff π L φ and π L ψ

π L φ∨ψ iff π L φ or π L ψ

π L φ→ψ iff π L φ implies π L ψ

π L ○φ iff π1 L φ

π L ◇φ iff there exists i≥0 such that πi L φ

π L □φ iff for all i≥0 we have πi L ○φ

π L φ∪ψ iff there exists i≥0 such that πi L ψ and for all j ∈ {0, 1, … i-1} we have πj L φ

is called the satisfaction relation. It is a relation between formulas and infinite sequences of states in the presence of a state labeling with atom sets.

When the labeling L is fixed, we can write π φ instead of π L φ

도표, 라인, 원, 화이트이(가) 표시된 사진

자동 생성된 설명

이런식임.

**6. Labeled Transition Systems**

> A labeled transition system(LTS for short) is a triple M = (S, →, L) consisting of

- S a finite set of stated

- → ⊆ S x S a transition relation

- L : S → P(Atoms) a labeling function

such that every state has an outward transition, i.e., for all s1 ∈ S there exists s2 ∈ S with s1→s2.

A path π is an LTS M = (S, →, L) is an infinite sequence of states s0, s1, s2, … such that for all i≥0, si →si+1.

Paths are written as π = s0→ s1→ s2→ s3→

> Example

- Recall the example with two parallel processes, where, for i∈ {1, 2}

ni denotes “process i **n**ot in critical section”

ri denoted “process i **r**equesting to enter critical section”

ci denoted “process i in **c**ritical section”

Atoms = {n1, n2, r1, r2, c1, c2}

텍스트, 도표, 라인, 원이(가) 표시된 사진

자동 생성된 설명

> Visualize all paths from a given state a by unwinding the LTS(Labeled Transition System) to obtain an infinite tree. (LTS를 풀어 infinit하게 전개하여 시작 상태에서 가능한 모든 경로를 가지치기하여 표현한 것.)

도표, 원, 라인이(가) 표시된 사진

자동 생성된 설명

Figure 1. LTS example

All possible path starting in s0 :

(s0 → s1 →)∞

(s0 → s1 →)n (s2 →)∞

(s0 → s1 →)n → s0 → (s2 →)∞

**7. Semantics for Labeled Transition Systems**

> Definition

Let M = (S, →, L) be an LTS and φ be and LTL formula.

We extend the satisfaction relation from infinite sequences to LTS’s as follows

: For a state s∈S, we define M, sφ, read M satisfies φ in state s or φ holds for M state s, to mean that πLφ for every path π of M starting at state s.

>Example(Figure1 참고)

1. M, s0p∧q

2. M, s0¬r

3. M, s0○r

4. M, s0○(q∧r)

5. M, s0□¬(p∧r)

6. M, s2□r

7. M, s0◇(¬q∧r)→◇□r

8. M, s0□◇p

9. M, s0□◇p→□◇r

10. M, s0□◇r→□◇p

> Property

- The safety property : Only one process may execute critical section code at any point

- The liveness property : Whenever a process requests to enter its critical section, it will eventually be allowed to do so

- The non-blocking property : A process can always request to enter its critical section

**8. Formula Equivalence**

> Two formulas φ and ψ are equivalent, denoted φ≡ψ, if they are satisfied by(i.e., hold for) exactly the same state labelings and infinite sequences of stated :

Given any labeling L : S→P(Atoms) and any infinite sequence of states π, we have that πL φ iff πLψ, in other words :

1) πL φ implies πL ψ

2) πL ψ implies πL φ

> Propositional tautologies

¬(φ∧ψ)≡¬φ∨¬ψ ¬(φ∨ψ)≡¬φ∧¬ψ

> Duality laws

¬○φ≡○¬φ ¬□φ≡◇¬φ ¬◇φ≡□¬φ

> Distributive laws

□(φ∧ψ)≡□φ∧□ψ ◇(φ∨ψ)≡◇φ∨◇ψ ○(φ∪ψ)≡○φ∪○ψ

> Inter-definalbility laws

◇φ≡¬□¬φ □φ≡¬◇¬φ ◇φ≡Т∪φ

Where Т(read “True”) is an abbreviation for p→p for some atom p

> Idempotency laws

◇◇φ≡◇φ □□φ≡□φ (φ∪ψ)∪ψ≡φ∪ψ φ∪(φ∪ψ)≡φ∪ψ

> Absorption laws

□◇□φ≡◇□φ ◇□◇φ≡□◇φ

> Expansion laws

◇φ≡φ∨○◇φ □φ≡φ∧○□φ φ∪ψ≡ψ∨(φ∧○(φ∪ψ))