TEMPORAL LOGIC

**Static Knowledge**

With all the languages seen so far, knowledge is considered static

This is meaningful for describing many domains but insufficient in the context of processes

We need a way to deal with time and dynamic (that changes) knowledge

**Time**

To handle time, we must first decide its nature

We have to decide how to represent time.

Is it:

• discrete or continuous? counting time as day or something continuous

• linear or branching? branching as different possibilities

• unidirectional or bidirectional? Can we express things about the past? *A product will be shipped only if it was payed*

• bounded, finite, or infinite? Know that the process will finish

Each choice requires a different strategy also a different representation and a different strategy of how do handle it

We will have to decide at each time what knowledge we have

**LTLf**

f stands for “Finite”

Simple logic much used

To showcase the main ideas, we study LTLf

A temporal logic with:

• discrete time

• linear evolution (no branching)

• considering only the future

• finite (but unbounded) time

LTLf extends propositional logic with “next” (⚪)and “until” ( ų ) operators

Two constructor

*In the next point of time something will happen* ⚪

*Something will happen until something else happen* ų

**Syntax of LTLf**

LTLf formulas are built through the grammar rule

φ::= x | ¬φ | φ ∧ φ | ⚪ φ | φ ų φ

• ⚪ next: in the next point in time, φ is true

• φ ų ψ: φ is true until ψ becomes true ψ at some point it must become true

*The product is not shipped until the product is payed*

**Semantics of LTLf**

To interpret LTLf formulas, we consider a finite sequence of timepoints {0, 1, . . . , n}

Sequence of point at each point tell what is true

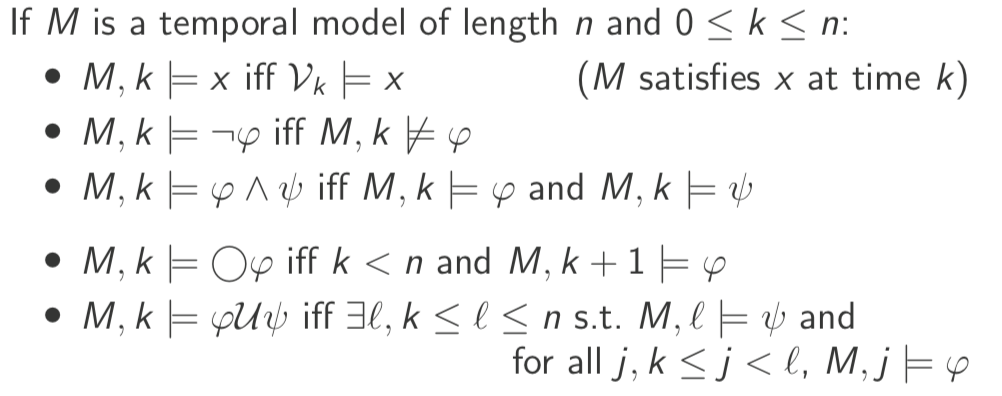
timepoint 0 is the present

At each timepoint, there is a propositional valuation

**Semantics of LTLf (II)**

A temporal model is a finite sequence M = V0,...,Vn of propositional valuations. We have n+1 valuations

In this case, n is the length of the temporal model M



M: V0 {x, ¬ y} V1{x,y} V2{ ¬ x, ¬ y}

M,1 ⊨ x

M,1 ⊨ y

M,0 ⊭ y

M, 0 ⊨ ¬ y

My last time point can’t have ⚪ (because there is nothing after that)

M,1 ⊨ x ^ y

M, 0 ⊨ ⚪ (x^y)

M,2 ⊭ ⚪ x (don’t satisfy any ⚪ )

φ ų ψ Eventually ψ at some point will become true, in between φ is true

M,1 ⊨ x ų ¬ y

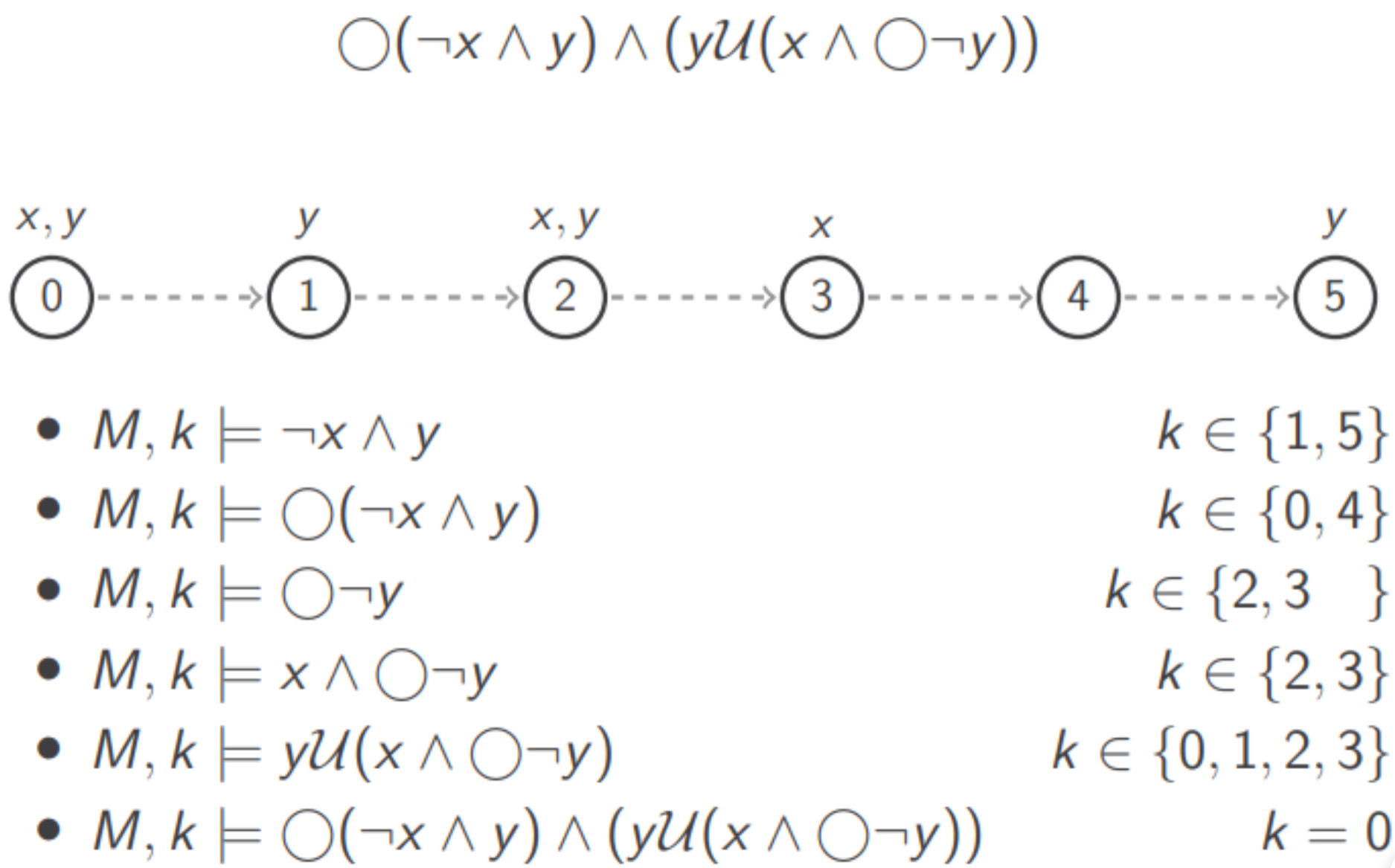
M,0 ⊨ x ų ¬ y

**Semantics of LTLf (III)**

The temporal model M satisfies φ iff M, 0 ⊨ φ

The formula φ is satisfiable iff there is a temporal model that satisfies it

**Example**

****

This is a model of length 5

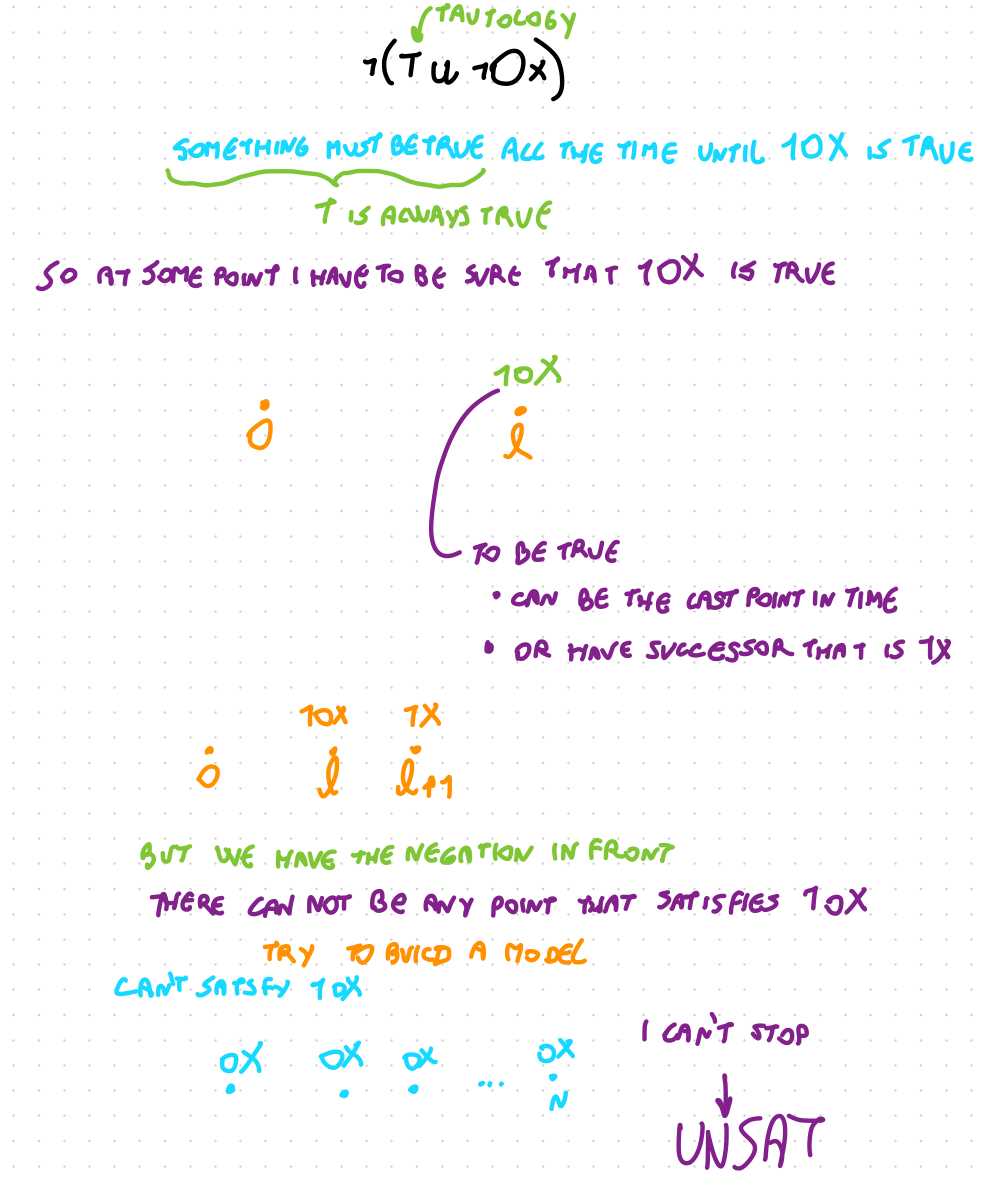
Only explicitly on the graph of what is true

* Check when ¬ x and y are both true is at point 1 and 5
* Next ¬ and y true, just subtract 1, so 0 and 4
* .
* Satisfy y until ( ) is true
* The present name true this one

**Example II**

¬(⊤ ų ¬ ⚪ x)

Ignore ¬ for a second



**Useful Abbreviations**

• ⊥ := x ∧ ¬x

• T := x v ¬x

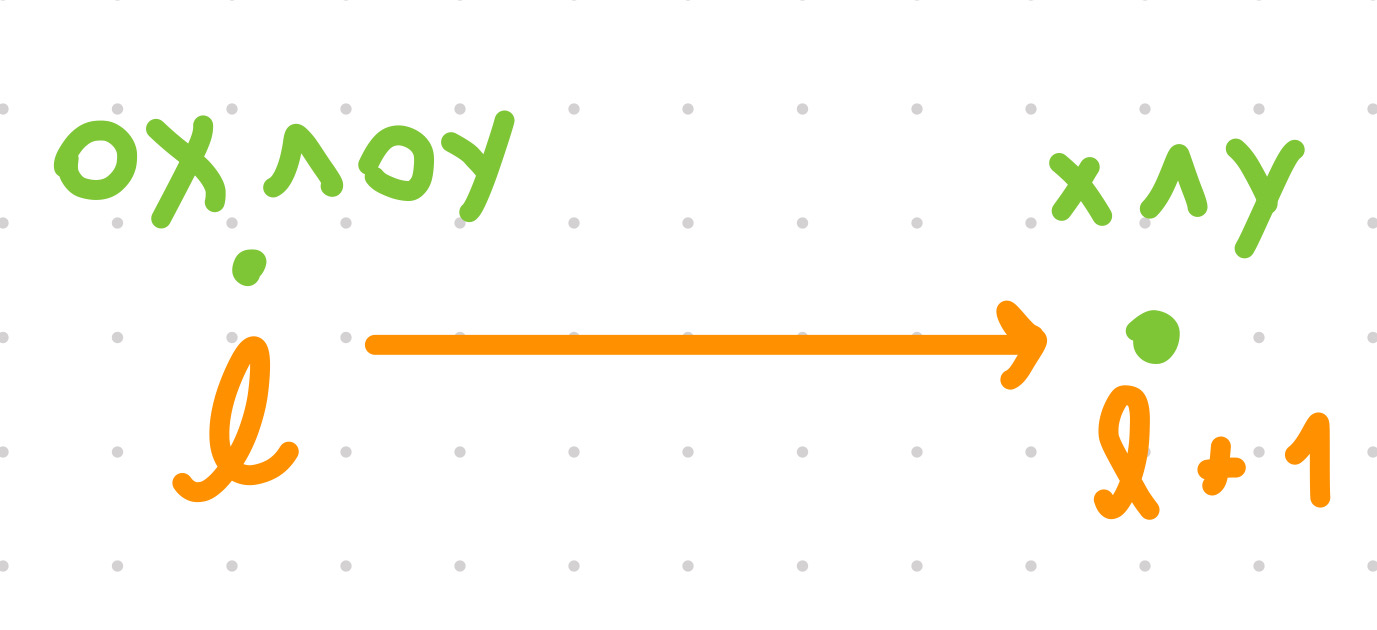
• ♢x := ⊤ ų x (eventually x) T will be true until x will be true, guaranteed to happen. ♢x at some point of the future x will be satisfied. Like an existential, there exists a time *n* in which x is satisfied. Like ∃x

• ☐x := ¬ ♢ ¬x (always x) x is always true if there is no time in the future when x is false. Like forall ∀x = ¬∃¬x

**Checking Satisfiability**

To check satisfiability, we could develop a tableaux-like procedure, decompose the formula in easier part

• Need to be careful with multiple ⚪ (only one temporal successor)



• Must find a way to handle ų

We present a different approach: a structure that describes all temporal models satisfying φ

**Understanding Until**

Recall what φ ų ψ means:

• there is a point in time (now or in the future) where ψ holds

• until that time arrives, φ is true

It all depends on when we satisfy ψ

Check when the second formula is true

Do we satisfy it now, or later?

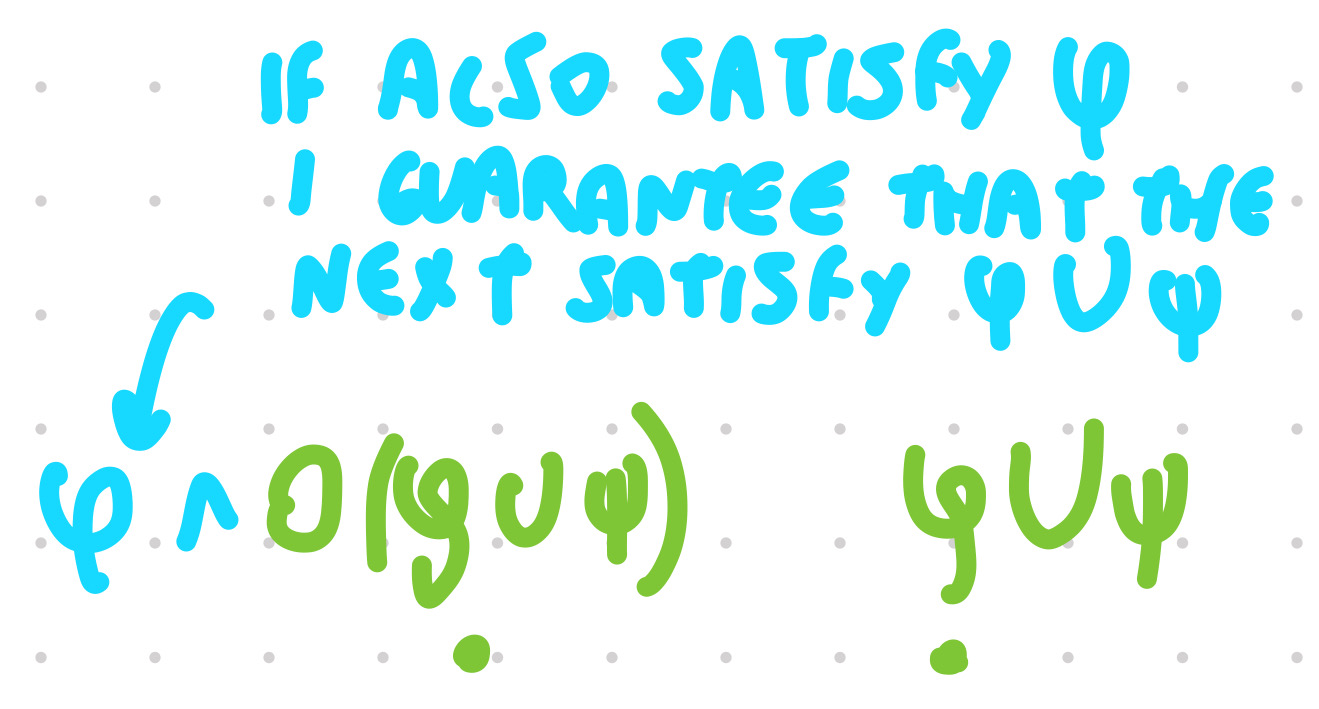
**Equivalence of Until**

For φ ų ψ, we can decide to satisfy it now (ψ)

or differ it to later ( ⚪ φ ų ψ)

In symbols,

a ų b and b ∨ (a ^ (⚪ a ų b )) are equivalent



**Temporal Model Characterisation**

A temporal model is a (finite) sequence of propositional valuations

These define (deterministically) which formulas are satisfied at each timepoint

We describe a temporal model as a sequence of sets of formulas (relevant)

At each point of time which formula is satisfied

**Temporal Model Characterisation II**

The temporal models satisfying φ share these properties:

• φ is satisfied at the first timepoint

Must be true

• at the last timepoint, no formula ⚪ψ is satisfied

When we finish we can’t satisfy a next formula

• at timepoint n, a formula ψ is satisfied iff ¬ψ is not satisfied

The set of formula has to be consistent, either the formula or its negation, can’t have both but at least 1

• ⚪ψ is satisfied at timepoint n iff ψ is satisfied at timepoint n + 1

• ψ1 ų ψ2 is satisfied at timepoint n iff ψ2 or ψ1⚪ψ1 ų ψ2 ^ ψ1 is satisfied at timepoint n

**Idea**

We use these properties to succinctly describe all temporal models

Using sequences of types full consistent sets of relevant formulas

**Subformulas and Closure**

The set sub(φ) of subformulas of φ is the smallest set s.t.

• φ ∈ sub(φ)

• if ψ1 ∧ ψ2 ∈ sub(φ) then {ψ1, ψ2} ⊆ sub(φ)

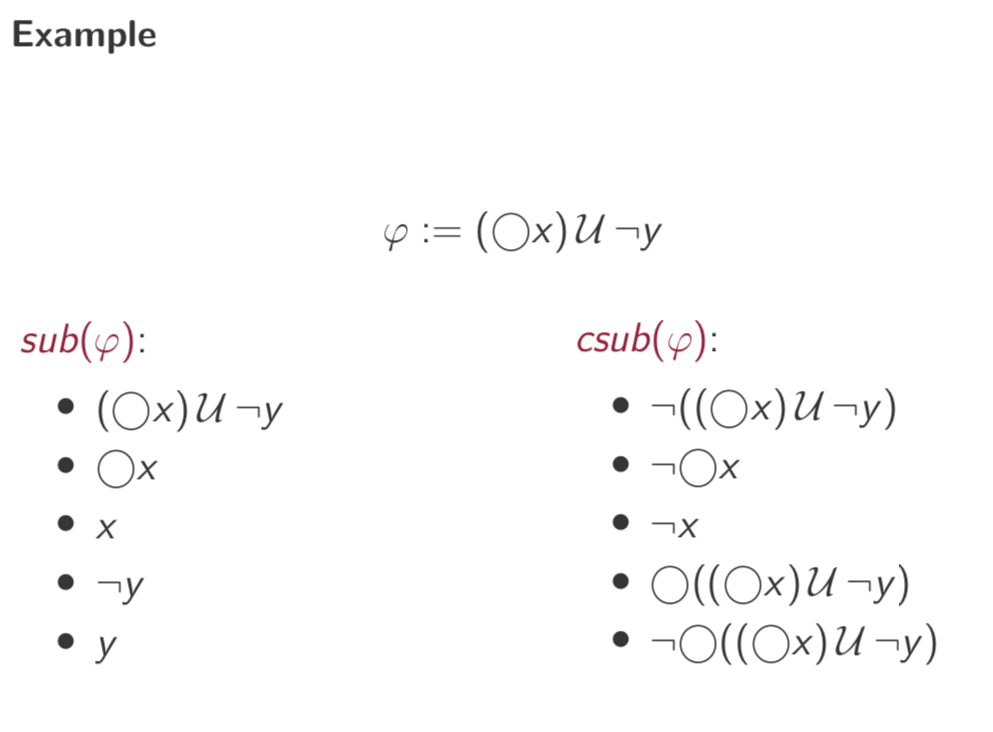
• if ¬ψ ∈ sub(φ) then ψ ∈ sub(φ)

• if ψ1 ų ψ2 ∈ sub(φ) then {ψ1 ,ψ2 } ⊆ sub(φ)

• if ⚪ψ ∈ sub(φ) then ψ ∈ sub(φ)

The closure of this set is

csub(φ) = {ψ, ¬ψ | ψ ∈ sub(φ)} ∪ {⚪ψ1 ų ψ2,¬⚪ψ1 ų ψ2 | ψ1 ų ψ2 ∈ sub(φ) }



subformula: all the pieces of the formula and the initial formula

closure: do the negation of all the formulas and since there is an until add the next and the negation of the next

**Types**

A type is a maximally consistent subset of csub

Specifically, *τ* is a type iff for every formula ψ ∈ csub(φ):

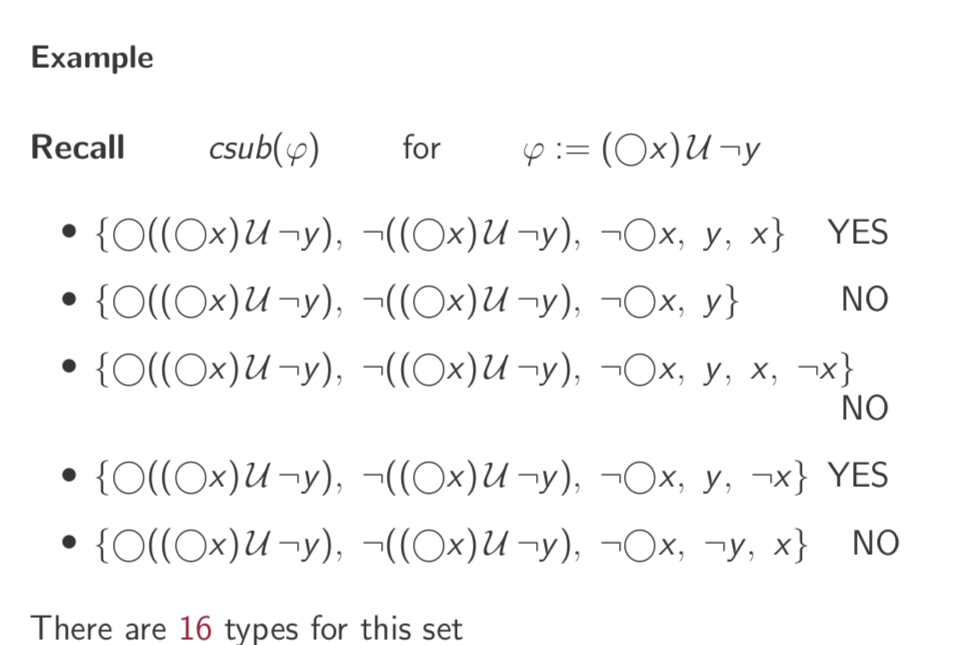
• {ψ, ¬ψ} ̸⊆ *τ* (consistency) don’t have both the formula and it’s negation

• {ψ, ¬ψ} ∩ *τ* ≠ ∅ (maximality) only one of the formula, not 0 or 2

• if ψ = ψ1 ∧ ψ2, then ψ ∈ *τ* iff {ψ1, ψ2} ⊆ *τ* (conjunctive consistency)

• if ψ = ψ1ų ψ2, then ψ ∈ *τ* iff either ψ2 ∈ *τ* or {ψ1, ⚪ψ} ⊆ *τ* ( ų -consistency) or ψ2 is true now or ψ1 is true and in the next point in time the until formula is true

A type is a set of subformulas of the closure of the subformulas that satisfied this properties (either the formulas or its negation and it is consistent with the conunctions and the untils)



The second one is not a type because it is missing x or ¬x

The third one is not a type because it is not consistent we have both x and ¬x

The last one we have ¬y, then the until is also true, not consistent with the semantic of until, so it is not a type

32 maximally consistent subset, 5 fundamental formulas can be true or false (2^5) = 32

**From Types to Valuations**

Importantly, every type τ defines exactly one valuation Vτ

But the converse is not true:

there exist *τ*1 ≠ *τ*2 s.t. V*τ*1 = V*τ*2

**Graph of Types**

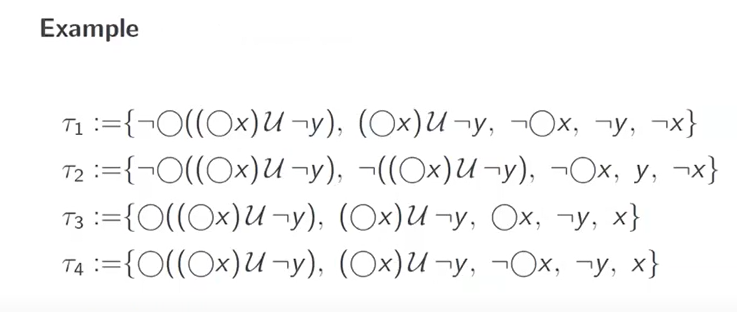
The graph of types of φ is the directed graph Gφ such that:

• the nodes of Gφ are the types of φ

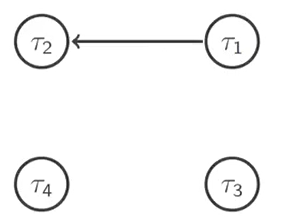
• there is an edge from *τ*1 to *τ*2 iff for every ⚪ψ∈csub(φ): ⚪ψ∈τ1 iff ψ ∈τ2

(possible time transitions)

**Example**



This is a part of the graph with 4 types

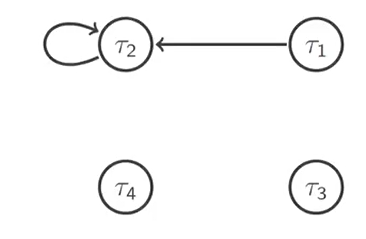


*τ*1 is connected to *τ*2

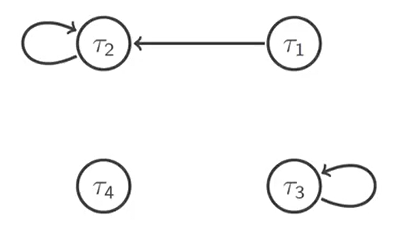
*τ*1 ¬⚪((⚪x) ų ¬y) also have ¬⚪x

*τ*2 ¬((⚪x) ų ¬y) and *τ*2 have ¬x

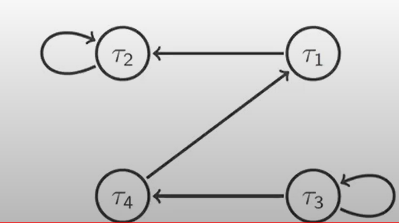
*τ*1 can’t be connected to *τ*3 or *τ*4 because they both have ((⚪x) ų ¬y)



*τ*2 ¬⚪((⚪x) ų ¬y) is connected to itself, can’t go to τ1, τ3 or τ4 because they have *τ*1 ⚪((⚪x) ų ¬y)



τ3 can be connected to itself, for the same reason of τ2



The graph is incomplete!

τ2 is a good place to finish because no next

**Paths**

Paths in the graph of types describe a “valid” behaviour in a temporal model

But to be in fact a temporal model (satisfying φ) we need two more conditions

• the path should start with a type containing φ at the beginning 0

• the path should end without a ⚪ hanging, must be a finite path

25 nov 2021

Building a structure that represent all the models that satisfy the formula

a type is the maximally consistent sets of relevant formulas

**Initial and Final Types**

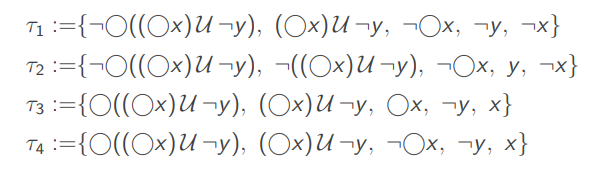
A type τ is:

• initial if φ ∈ τ if it contains the formulas we are interested in

• final if it contains no formula of the form ⚪ ψ no next operator, can contain ¬⚪

Type: maximally consistent set of relevant formulas (is the closure of the subformulas including the negation, the next and the until formulas)

**Example**



(actually there are 16 cases of type in total)

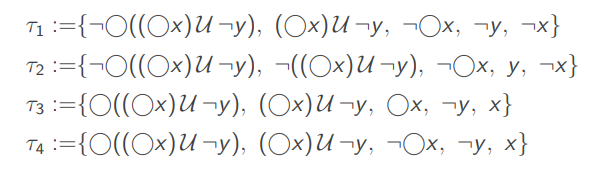
• initial: τ1, τ3, τ4 they contain the formula we are interested in (⚪x) ų ¬y

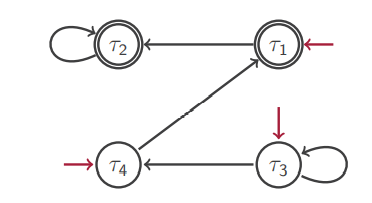
• final: τ1, τ2 do not contain any (positive) next formula

In general, a type may be initial and final or none (not in this example)

those property do **not** exclude each other

**Graphical Representation**





initial type: incoming arrow, coming from nowhere

final type: double circle

**Temporal Models**

There is a one-to-one correspondence between temporal models satisfying φ and paths in Gφ from an initial to a final type can be more or less long, can loop it as long as I want but at some point I have to reach τ1 or τ2 (the final)

If I have a temporal model I can build a path

For a path τ1, τ2, . . . , τn, the sequence Vτ1 , . . . , Vτn is a temporal model satisfying φ

to prove: at each time n it satisfy all the formulas in τn

Given a temporal model M = V1, . . . , Vn and 1 ≤ k ≤ n, let τi := {ψ ∈ csub(φ) | M, k ⊨ ψ}

Then τ1, . . . , τn is a path of Gφ

**Sidenote**

The graph that we built with initial and final nodes is called an **automaton**

accept the language of all the temporal models

Automaton (field of research) are often used for reasoning in different logics and specialised techniques have been developed

We will not delve deeper into this topic here

**How is LTLf used?**

describes processes

LTLf is often used to describe processes

• the protocols of an online store

• the switching of traffic lights

• the scheduler in an operating system

• . . .

Satisfiability allows us to verify their properties

verify that have some specific properties

• are products shipped only after being paid? is the design of the system such that it can never be shipped before it is paid

• may two intersecting roads have both a green light? make sure that the protocol is that at least one is red

• is every request eventually granted?

• . . .

**Guarantees and Possibilities**

Let φ describe a process and ψ a property

• ψ is guaranteed by φ iff φ ⇒ ψ iff φ ∧ ¬ψ is **unsatisfiable**

A model φ ⇒ ψ is that same of saying that is impossible to build a model that satisfy φ ∧ ¬ψ

• ψ is possible in φ iff φ ∧ ψ is **satisfiable**

something is allowed in the system but not always

if is possible to pay with credit card, it is accept but also with cash

The latter can also be used to verify whether an execution complies with the specification

**Summary**

Literal temporal logical with finite time

We looked at LTLf as a simple example of temporal logic for representing and reasoning about temporal properties

As usual, many variants can be introduced, depending on the properties of the knowledge domain

The basic ideas and techniques are prototypical

**ALCLTLf**

The logic ALCLTLf extends LTLf  by providing evolving interpretations

A temporal interpretation is then a finite sequence of ALC interpretations

• a global TBox (must hold at every point in time)

• ABox with temporal operators; potentially timestamps

Of course, several technicalities to consider

semi rigid: once you are a parent you can not stop being a parent

you can be hungry at one time and then stop, so is not rigid