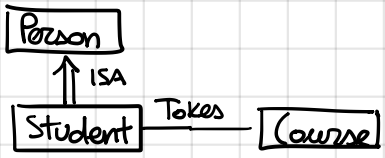
FIRST ORDER LOGIC

FOL is the logic to speak about objects which are the domain of universe, concerned about properties and relationships. It also has functions including constants that denote objects.



Student(x) = "x is a student"
Person(x)
Course(x)
Takes(x,y) = "x takes y"
Names Arities

This is our kind of alphabet!

$\forall x \ St(x) \rightarrow Pe(x)$ [ISA]
$\forall x (\exists y \ Takes(x,y)) \rightarrow St(x)$ [First argument of "Takes" is a student]
$\alpha(x)$
$\forall y (\exists y \ Takes(x,y)) \rightarrow Co(y)$ [Second argument of "Takes" is a Course]
$\alpha(y)$
$\Rightarrow \forall x \forall y \ Takes(x,y) \rightarrow St(x) \wedge Co(y)$
$\beta(x,y)$

Note: first 3 formulas are talking in general, not about any particular object (Student, Person or Course). Just Metadata → INTENSIONAL KNOWLEDGE

Student(John)
Takes(John, Moth)
Person(Mary)
} Γ (= gamma)

Constants → Data → EXTENSIONAL KNOWLEDGE

↳ $\Gamma \models Pe(John)$ ✓ [Is John a person?]
$\Gamma \models Co(Moth)$ ✓
$\Gamma \models St(Mary)$ ×
$\Gamma \models \neg St(Mary)$ ×
} Because general rules can't lead to these two conclusions

Vars = $\{x_1 \dots x_n\}$ set of individual variables. Vars $\subseteq$ Terms eg constants
FOL interpretation: $I = (\Delta^I, P_1^I, P_2^I \dots f_1^I, f_2^I \dots)$ P_x^I = predicate f_x^I = formula
- Δ^I = domain, set of objects

Given I, an assignment is a function $\alpha : Vars \rightarrow \Delta^I$ that assigns to each variable $x \in Vars$ an object $\alpha(x) \in \Delta^I$ → $\hat{\alpha} : Terms \rightarrow \Delta^I$
- $\hat{\alpha}(x) = \alpha(x)$ if $x \in Vars$
- $\hat{\alpha}(f(t_1 \dots t_k)) = f^I(\hat{\alpha}(t_1) \dots \hat{\alpha}(t_k))$

$I, \alpha \models \varphi$: φ is true in interpretation I wrt an assignment α

Variable x in formula φ is free if x doesn't occur in scope of any quantifier, bounded otherwise. φ is closed if has no free variable (sentence), open otherwise

A FOL query is an open formula. When φ is a query with free variables $(x_1 \dots x_k)$ then we write it as $\varphi(x_1 \dots x_k)$ and say that φ has arity k.

Boolean query is a FOL query without free variables

LOGICAL TASKS

Validity: φ is valid iff $\forall I, \alpha$ we have $I, \alpha \models \varphi$

Satisfiability: φ is satisfiable iff $\exists I, \alpha$ s.t. $I, \alpha \models \varphi$, unsatisfiable otherwise

Logical Implication: φ logically implies ψ $(\varphi \models \psi)$ iff $\forall I, \alpha$ if $I, \alpha \models \varphi$ then $I, \alpha \models \psi$

Logical Equivalence: φ is logical equivalent to ψ iff $\forall I, \alpha$ we have that $I, \alpha \models \varphi$ iff $I, \alpha \models \psi$ i.e $\varphi \models \psi$ and $\psi \models \varphi$

QUERY EVALUATION

Checking if a formula is true given interpretation and assignement. To do so we have to consider finite alphabet (finite predicates, functions) and finite interpretation (Δ^I finite)

$\Rightarrow$ Given query $\varphi(x_1 \dots x_k)$ compute $\varphi^I = \{(a_1 \dots a_k) \mid I, \langle a_1 \dots a_k \rangle \models \varphi(x_1 \dots x_k)\}$

Recognition problem: given finite I, a query and a tuple, check is the tuple is a solution.

Combined Complexity: complexity of $\{\langle I, \alpha, \varphi \rangle \mid I, \alpha \models \varphi\}$ i.e I, α, φ all part of input

Data Complexity: All but φ (fixed) is part of input

Query Complexity: All but I (fixed) is part of input

Theorem: Complexity of $\{\langle I, \alpha \rangle \mid I, \alpha \models \varphi\}$ is

- TIME: polynomial
- SPACE: LogSpace

This is the reason why FOL queries are used in databases!

Example

We consider FOL interpretations exactly as used in relational databases. This requires to drop functions except for constants. Moreover we assume that the interpretation of constants is the identity function, that is constants are interpreted as themselves. This allows us to drop also the interpretation of constants from our interpretations, which now have the form:

$$\mathcal{I} = (\Delta^{\mathcal{I}}, P_1^{\mathcal{I}}, P_2^{\mathcal{I}}, \dots, P_n^{\mathcal{I}}).$$

Interpretation: $\mathcal{I}$ is as follows (also given in relational notation):

- $\Delta^{\mathcal{I}} = \{john, paul, george, mick, ny, london, 0, 1, \dots, 100\}$
- $Person^{\mathcal{I}} = \{(john, 30), (paul, 60), (george, 35), (mick, 35)\}$
- $Lives^{\mathcal{I}} = \{(john, ny), (paul, ny), (george, london), (mick, london)\}$
- $Manages^{\mathcal{I}} = \{(paul, john), (george, mick), (paul, mick)\}$

FOL ↕ DATABASE

$Person^{\mathcal{I}}$

name	age
john	30
paul	60
george	35
mick	35

$Lives^{\mathcal{I}}$

name	city
john	ny
paul	ny
george	london
mick	london

$Manages^{\mathcal{I}}$

boss	emp. name
paul	john
george	mick
paul	mick

Query: find name and age of persons who live in the same city as their boss.

$$\exists z, w.\text{Person}(x, y) \wedge \text{Manages}(z, x) \wedge \text{Lives}(x, w) \wedge \text{Lives}(z, w)$$

"Find" or "Return" mean assign a value to open variable! → Query = open formula (as we said)

EXAMPLES

S(s_id, name) B(b_id, color) R(s_id, b_id, date)

① Find names of sailors who have reserved boat 103

$\exists x,z,w,t\ S(x,y) \wedge R(x,z,w) \wedge B(z,t) \wedge z=103$

↳ this means we ONLY CARE of y

↳ i.e "whatever x,z,w,t"

↳ needed to "force" z to be a boat id and not just a number

② find names of sailors who have reserved a red boat

$\exists sid,bid,d\ S(sid,n) \wedge R(sid,bid,d) \wedge B(bid,"red")$

③ Find colors of boat reserved by BOB

$\exists bid,sid,d\ B(bid,c) \wedge R(sid,bid,d) \wedge S(sid,"BOB")$

④ Find names of sailors who have reserved at least one boat

$\exists sid,bid,d,c\ S(sid,n) \wedge R(sid,bid,d) \wedge B(bid,c)$

⑤ Find names of sailors who reserved red and green

$\exists sid,bid,d,bid_2,d_2\ S(sid,n) \wedge R(sid,bid,d) \wedge B(bid,"RED") \wedge R(sid,bid_2,d_2) \wedge B(bid_2,"GREEN")$

⑥ ... red OR green

$\exists sid,bid,d\ S(sid,n) \wedge R(sid,bid,d) \wedge (B(bid,"RED") \text{ OR } B(bid,"GREEN"))$

⑦ ... reserved AT LEAST 2 boats

$\exists sid,bid,bid_2 \ldots\ S(sid,n) \wedge R(sid,bid,d) \wedge B(bid,c) \wedge R(sid,bid_2,d_2) \wedge B(bid_2,c_2) \wedge \neg(bid=bid_2)$

⑧ ... not reserved red boats

$\exists x\ S(x,n) \wedge \neg\exists y,d\ R(x,y,d) \wedge B(y,"RED")$

⑨ ... reserved ALL boat

$\exists x\ S(x,n) \wedge \forall y\,(\exists c\ B(y,c)) \rightarrow (\exists d\ R(x,y,d))$

⑩ ... all red boats

$\exists x\ S(x,n) \wedge \forall y\ B(y,"red") \rightarrow \exists d\ R(x,y,d)$

Imply

$\rightarrow = \supset$

⑪ ... only red boats

$\exists x\ S(x,n) \wedge \forall y\ \exists d\ R(x,y,d) \rightarrow B(y,"red")$

⑫ ... exactly one boat

$\exists x\ S(x,n) \wedge (\exists y,d,c\ R(x,y,d) \wedge B(y,c)) \wedge (\forall yy'(\exists d\ R(x,y,d)) \wedge (\exists d\ R(x,y',d) \supset y=y'))$

TABLEAU METHOD

Proving in mechanical manner that a given set of formulas is not satisfiable

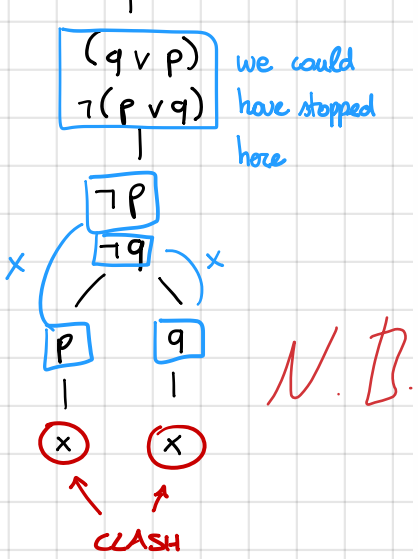
given : set of premises Γ and conclusion ϕ

task : prove $\Gamma \models \phi$

how : show $\Gamma \cup \neg\phi$ is not satisfiable i.e. add the complement of the conclusion to the premises and derive a contradiction

Th. $\Gamma \models \phi$ iff $\Gamma \cup \{\neg\phi\}$ is unsatisfiable

eg. $\neg(q \vee p \supset p \vee q)$



N.B.

α rules

$$\frac{\phi \wedge \psi}{\phi \quad \psi} \qquad \frac{\neg(\phi \vee \psi)}{\neg\phi \quad \neg\psi} \qquad \frac{\neg(\phi \supset \psi)}{\phi \quad \neg\psi}$$

$\neg\neg$-Elimination

$$\frac{\neg\neg\phi}{\phi}$$

β rules

$$\frac{\phi \vee \psi}{\phi \mid \psi} \qquad \frac{\neg(\phi \wedge \psi)}{\neg\phi \mid \neg\psi} \qquad \frac{\phi \supset \psi}{\neg\phi \mid \psi}$$

Branch Closure

$$\frac{\phi \quad \neg\phi}{X}$$

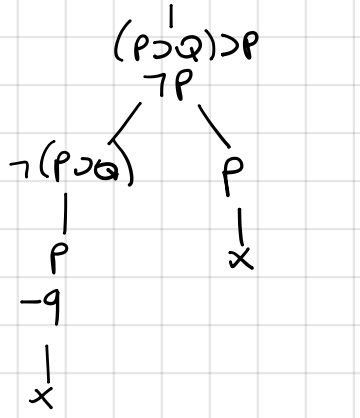
Do α before β always

Note: These are the standard ("Smullyan-style") tableau rules.

We omit the rules for $\equiv$. ~~We~~ rewrite $\phi \equiv \psi$ as $(\phi \supset \psi) \wedge (\psi \supset \phi)$

Eg $\models ((P \supset Q) \supset P) \supset P$

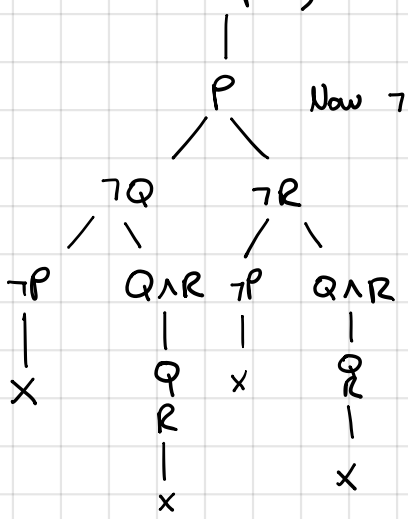
$\neg(((P \supset Q) \supset P) \supset P)$



Eg $P \supset (Q \wedge R), \neg Q \vee \neg R \models \neg P$

$P \supset Q \wedge R$
$\neg Q \vee \neg R$
$\neg\neg P$

In a "node" with multiple formulas we compute one formula at time (now $\neg\neg P$)



Now $\neg Q \vee \neg R$

Now $P \supset Q \wedge R$ for each node

Note : order is ininfluent eg $q \vee p$ = $p \mid q$ or $q \mid p$

Note : $\models \ldots$ becomes $\neg\neg \ldots$

$\neg(P \vee Q \supset P \wedge Q)$ is satisfiable?

$P \vee Q$

$\neg(P \wedge Q)$

P | Q

$\neg P$ | $\neg Q$ | $\neg P$ | $\neg Q$

x | 0 | 0 | x

yes, it is

The Tableau shows us all possible interpretations ($\{P\}, \{Q\}$) that satisfy the formula.
eg $I = \{P\}$ means only P is true

Th. Soundness
If $\Gamma \vdash$ (proves) ϕ then $\Gamma \models \phi$

Th. Completeness
If $\Gamma \models \phi$ then $\Gamma \vdash \phi$

Other rules for propositional logic (β rules)

① $\dfrac{\phi \equiv \psi}{\begin{array}{c|c}\phi & \neg\phi \\ \psi & \neg\psi\end{array}}$ ② $\dfrac{\neg(\phi \equiv \psi)}{\begin{array}{c|c}\phi & \neg\phi \\ \neg\psi & \psi\end{array}}$

δ rules:

① $\dfrac{\exists x\ \phi(x)}{\phi(c)}$ ② $\dfrac{\neg\forall x\ \phi(x)}{\neg\phi(c)}$

c = fresh constant = new constant not previously appearing in tableaux

γ rules:

① $\dfrac{\neg\exists x\ \phi(x)}{\neg\phi(t)}$ ② $\boxed{\dfrac{\forall x\ \phi(x)}{\phi(t)}}$

t = any term (not fresh)

This means that for every object of the domain the property $\phi(x)$ should be true. A term t that occurs in the tableaux denotes an object of the domain.
Therefore ϕ must be true for all terms t in the tableaux i.e. $\forall$ can be applied as many times as one want to any term.

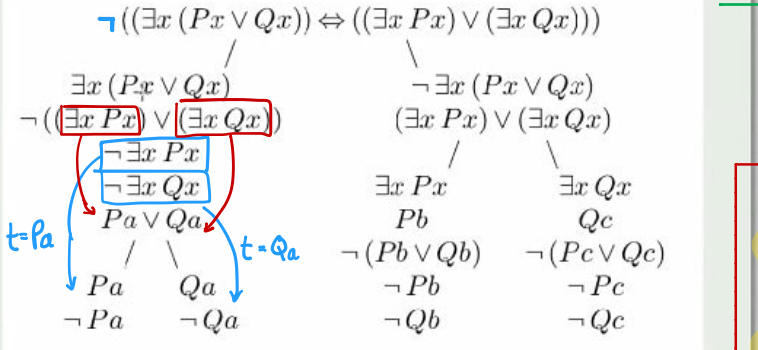
Notation: $\phi[x/t]$ = formula we get by substitute all x with t

eg $\forall x P(x,y)[x/b] = \forall x P(x,y)$
$P(x,y,f(x))[x/a] = P(a,y,f(a))$
$\exists x P(x,x) \wedge Q(x)[x/c] = \exists x P(x,x) \wedge Q(c)$
$\forall x P(x,y)[y/f(x)]$ = not allowed, x bounded to "$\forall$"

Example

To check if the formula $(\exists x(P(x) \vee Q(x))) \equiv ((\exists x P(x)) \vee (\exists x Q(x)))$ is ~~satisfiable~~ valid, we start with a tableaux with this formula:

Tableaux only do UNSAT but...



We have closhes → UNSAT
⇒ original formula is valid

GENERAL RULE
- is valid? → negate and check if all branches are closhed
- is satisfiable? → check if there is at least one open branch.

Ex. check if $\forall x P(x) \wedge \exists x \neg P(f(x))$ is SAT

$\forall x P(x)$

$\exists x \neg P(f(x))$

$\neg P(f(c))$ — fresh

$P(f(c))$

×

Note: in cases like this, start with $\exists$ and then use $\forall$ to find clashes.

Eg. $\forall$ applied multiple times

$\exists x.P(x) \wedge \forall x(P(x) \supset P(f(x)))$

$\exists x.P(x)$

$\forall x(P(x) \supset P(f(x)))$

$P(a)$

$P(a) \supset P(f(a))$ (1)

$\neg P(a)$ — CLASH

$P(f(a))$

$P(f(a)) \supset P(f(f(a)))$ (2)

$\neg P(f(a))$ — CLASH

$P(f(f(a)))$ ⋮

Understand the model:

$\exists x(P(x) \wedge \neg Q(x)) \wedge \forall y(P(y) \vee Q(y))$

$\exists x(P(x) \wedge \neg Q(x))$
$\forall y(P(y) \vee Q(y))$

$P(a) \wedge \neg Q(a)$ | $P(a) \vee Q(a)$

$P(a)$, $\neg Q(a)$ | $P(a)$ | $Q(a)$

OPEN | CLASH

From the formulas appearing in the OPEN branch of the tableaux it is possible to construct a model for the root formula.

- $\Delta = \{a\}$, the constants appearing in the formulas
- $I(P) = \{a\}$, since the formula $P(a)$ appears in the open branch
- $I(Q) = \{\}$ since the formula $\neg Q(a)$ appears in the open branch

CONJUNCTIVE QUERIES

A query of form $\exists \vec{y}\, conj(\vec{x}, \vec{y})$ where "conj" is a conjunction ("and") of atoms and EQUALITIES over free variable $\vec{x}$, the existentially quantified variable $\vec{y}$ and possibly constants.

Relational alphabet:
Person(name, age), Lives(person, city), Manages(boss, employee)

Query: find the name and the age of the persons who live in the same city as their boss.

$\Rightarrow$ $\exists b, e, p_1, c_1, p_2, c_2.\text{Person}(n, a) \wedge \text{Manages}(b, e) \wedge \text{Lives}(p1, c1) \wedge \text{Lives}(p2, c2) \wedge n = p1 \wedge n = e \wedge b = p2 \wedge c1 = c2$

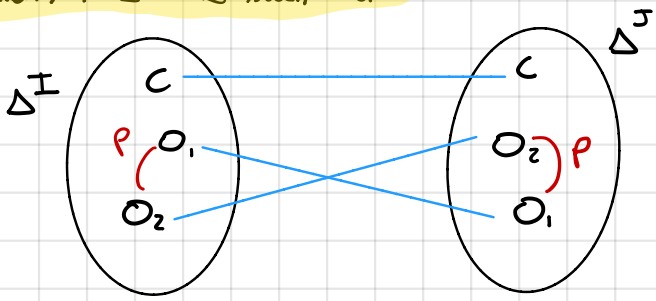
Or simpler: $\exists b, c.\text{Person}(n, a) \wedge \text{Manages}(b, n) \wedge \text{Lives}(n, c) \wedge \text{Lives}(b, c)$

HOMOMORPHISM

Suppose two interpretations I and J on some alphabet.

A homomorphism from I to J is a mapping (function) $h: \Delta^I \to \Delta^J$ such that:

- $h(c^I) = c^J$
- $(o_1 \dots o_k) \in P^I$ implies $(h(o_1) \dots h(o_k)) \in P^J$



If the map cover the entire domain then it is called isomorphism. FOL is unable to distinguish between interpretations that are isomorphic.

Eg $\Delta^I = \{v_1, v_2, v_3, v_4, v_5\}$

$e^I = \{(v_1, v_2), (v_2, v_3), (v_2, v_4), (v_4, v_3), (v_4, v_5)\}$

$a^I = v_1$

$b^I = v_3$

$\Delta^J = \{q_1, q_2, q_3\}$

$e^J = \{(q_1, q_2), (q_2, q_3), (q_3, q_3)\}$

$a^J = q_1$

$b^J = q_3$

We want $h(v_1)$, $h(v_2)$ etc

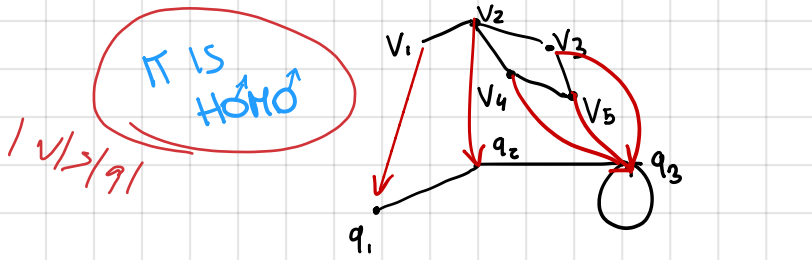
$h(v_1) = q_1$ — because $h(c^I) = c^J \to h(a^I) = a^J \to h(v_1) = q_1$ (same for b^I / b^J)

$h(v_2) = q_2$ — because $(v_1, v_2) \in e^I \Rightarrow (h(v_1), h(v_2)) \in e^J$ i.e we check e^J and search q_1 in first position

$h(v_3) = q_3$

$h(v_4) = q_3$ again q_3 like above

$h(v_5) = q_3$ "



CANONICAL INTERPRETATION

Given a (boolean) CQ eg $q(c) \leftarrow E(c,y).E(y,z).E(z,c)$

equal to $\exists y z \; E(c,y) \wedge E(y,z) \wedge E(z,c)$ can be transformed into a canonical interpretation

$$I_q = \begin{cases} \Delta^{I_q} = \{c, y, z\} \\ c^{I_q} = c \\ E^{I_q} = \{(c,y), (y,z), (z,c)\} \end{cases}$$

Def.: The canonical interpretation $\mathcal{I}_q$ associated with q is the interpretation $\mathcal{I}_q = (\Delta^{\mathcal{I}_q}, P^{\mathcal{I}_q}, \dots, c^{\mathcal{I}_q}, \dots)$, where

- $\Delta^{\mathcal{I}_q} = \{x_1, \dots, x_n\} \cup \{c \mid c \text{ constant occurring in } q\}$, i.e., all the variables and constants in q;
- $c^{\mathcal{I}_q} = c$, for each constant c in q;
- $(t_1, \dots, t_k) \in P^{\mathcal{I}_q}$ iff the atom $P(t_1, \dots, t_k)$ occurs in q.

Th. For boolean CQs, $I \models$ iff there exists a homomorphism from I_q to I

Observations:

① $I_q \models q$ always true.

② If h - homomorphism from h I to I_2 and h' from I_2 to I_3 then $h \circ h'$ is an omomorphism from I to I_3

Two interpretations I and J are **homomorphically equivalent** if exists homomorphism $h_{I,J}$ from I to J and h_{JI} from J to I.

CQs are unable to distinguish between interpretations that are homomorphic equivalent.

QUERY CONTAINMENT: given two FOL queries φ on ψ of same arity, φ is contained in ψ, denoted $\varphi \subseteq \psi$ if $\forall$ interpretations I and assignements α we have that: $I, \alpha \models \varphi$ implies $I, \alpha \models \psi$

For FOL queries, query containment is undecible

$\forall \vec{x}\ q_1(\vec{x}) \rightarrow q_2(\vec{x})$ VALID

For CQs, $q_1(\vec{x}) \subseteq q_2(\vec{x})$ can be reduced to query evaluation (NP complete)

How:

1. Freeze the free variables i.e consider them as constants
   $\Rightarrow I_{\alpha,\vec{c}} \models q_1(\vec{c})$ implies $I_{\alpha,\vec{c}} \models q_2(\vec{c})$ $\forall I_{\alpha,\vec{c}}$ where $\vec{c}$ are new constants and $I_{\alpha,\vec{c}}$ extends I to new constants with $c^{I_{\alpha,\vec{c}}} = \alpha(x)$
2. Construct canonical interpretation $I_{q_1(\vec{c})}$ of CQ $q_1(\vec{c})$
3. Evaluate on $I_{q_1(\vec{c})}$ the CQ $q_2(\vec{c})$ i.e check if $I_{q_1(\vec{c})} \models q_2(\vec{c})$

Th. For CQs $q_1(\vec{x}) \subseteq q_2(\vec{x})$ iff $I_{q_1(\vec{c})} \models q_2(\vec{c})$ where $\vec{c}$ are new constants

Th. For CQs $I \models q$ iff $q_I \subseteq q$

Th. CQs containment is NP-complete

UNION OF CQ (UCQs)

UCQ has the form $\bigvee_{i=1\dots n} \exists \vec{y_i}.\ conj_i(\vec{x}, \vec{y_i})$ i.e an OR of conjunctive queries.

As a normal "$\vee$", $I, \alpha \models \vee(\dots)$ iff $I, \alpha\ \exists y_i conj_i(\vec{x}, \vec{y_i})$ for some i i.e at least one y_i satisfy

Also for UCQs we can have containment: $\{q_1 \dots q_k\} \subseteq \{q'_1 \dots q'_n\}$ iff for each q_i there is a q'_j s.t. $q_i \subseteq q'_j$

QUERY ANSWERING WITH INCOMPLETE INFORMATION

If we dont have complete information we cant do $D \models Q$ (evaluation)

$\Rightarrow \forall I, I \models D$ implies $I \models Q$ (logical implication) i.e for each model we must check if Q is true in it too

A common form in which D can be expressed is INCOMPLETE DATABASE / NAIVE TABLES in which we can also find labelled nulls (unknown values)

i.e different values of null

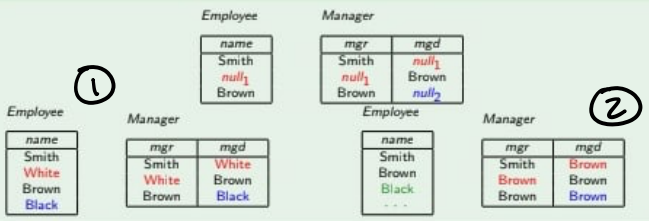
SQL cant use them!

Semantics of incomplete databases:

- A valuation function for nulls is a assignment function $\sigma : Nulls \to Const$ (essentially nulls are considered as individual variables in logic).
- We denote by $\mathcal{I}, \sigma \models D$ the fact that for every tuple $(t_1, \dots, t_n) \in P$ for each table P we have $\mathcal{I}, \sigma \models P(t_1, \dots, t_n)$.
- We define in logic the set of databases completing D as

$$Models(D) = \{\mathcal{I} \mid \text{there exists a } \sigma \text{ such that } \mathcal{I}, \sigma \models D\}$$

Example



Answering a query in incomplete databases means computing the certain answer denoted as $cert(q, \mathcal{D})$, the set of tuples $\vec{c}$ of constants of Const s.t. $\vec{c} \in q^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{D}$

- if q is boolean and $\mathcal{D}$ incomplete : $\mathcal{D} \models q$ iff q evaluates to true in every model $\mathcal{I}$ of $\mathcal{D}$ ($\mathcal{D} \not\models q$ otherwise)
- $\mathcal{D} \not\models q$ is totally different from $\mathcal{D} \models \neg q$

How to use CQs in incomplete DB?

- Each tuple in a table of $\mathcal{D}$ becomes an atom in conjunctive query $q_{\mathcal{D}}$
- Each labelled null occuring in $\mathcal{D}$ become an existentially quantified variable in $q_{\mathcal{D}}$

Th. $\mathcal{D} \models q$ iff $q_{\mathcal{D}} \subseteq q$ with q = boolean(union) of conjunctive queries.

Th. $\mathcal{D} \models q$ iff $I_{q_{\mathcal{D}}} \models q$ → nulls becomes constants ($x_1, x_2 \ldots$)

Example

E(mployee)

name
Smith
$null_1$
Brown

M(anager)

mgr	*mgd*
Smith	$null_1$
$null_1$	Brown
Brown	$null_2$

- Queries:
  $q_1(x, y) \leftarrow M(x, y)$
  $q_2(x) \leftarrow \exists y.M(x, y)$
  $q_3(x) \leftarrow \exists y_1, y_2, y_3.M(x, y_1) \wedge M(y_1, y_2) \wedge M(y_2, y_3)$
  $q_4(x, y_3) \leftarrow \exists y_1, y_2.M(x, y_1) \wedge M(y_1, y_2) \wedge M(y_2, y_3)$
- Answers:
  q_1: { }
  q_2: { Smith, Brown }
  q_3: { Smith }
  q_4: { }

For non boolean (U)CQs :

- evaluate q in $\mathcal{D}$ as it was a complete database
- filter all answers where nulls appears and remove them

UML CLASS DIAGRAM IN FOL

Lets stat with an exercise : → The specification of this domain can be done in UML

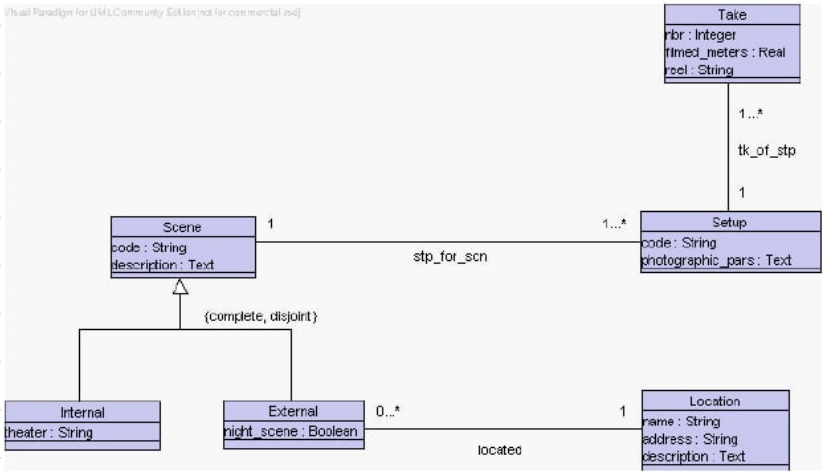
Requirements: We are interested in building a software application to manage filmed scenes for realizing a movie, by following the so-called "Hollywood Approach".

Every **scene** is identified by a code (a string) and it is described by a text in natural language.

Every scene is filmed from different positions (at least one), each of this is called a **setup**. Every setup is characterized by a code (a string) and a text in natural language where the photographic parameters are noted (e.g., aperture, exposure, focal length, filters, etc.). Note that a setup is related to a single scene.

For every setup, several **takes** may be filmed (at least one). Every take is characterized by a (positive) natural number, a real number representing the number of meters of film that have been used for shooting the take, and the code (a string) of the reel where the film is stored. Note that a take is associated to a single setup.

Scenes are divided into **internals** that are filmed in a theater, and **externals** that are filmed in a **location** and can either be "day scene" or "night scene". Locations are characterized by a code (a string) and the address of the location, and a text describing them in natural language.



All good but : not precise, verification done by humans, machine incomprehensible etc.

⇒ Use logic

Alphabet: $Scene(x)$, $Setup(x)$, $Take(x)$, $Internal(x)$, $External(x)$, $Location(x)$, $stp\_for\_scn(x, y)$, $ck\_of\_stp(x, y)$, $located(x, y)$, ...

Axioms:

$\forall x, y.\ code_{Scene}(x, y) \supset Scene(x) \wedge String(y)$
$\forall x, y.\ description(x, y) \supset Scene(x) \wedge Text(y)$

$\forall x, y.\ code_{Setup}(x, y) \supset Setup(x) \wedge String(y)$
$\forall x, y.\ photographic\_pars(x, y) \supset Setup(x) \wedge Text(y)$

$\forall x, y.\ nbr(x, y) \supset Take(x) \wedge Integer(y)$
$\forall x, y.\ filmed\_meters(x, y) \supset Take(x) \wedge Real(y)$
$\forall x, y.\ reel(x, y) \supset Take(x) \wedge String(y)$

$\forall x, y.\ theater(x, y) \supset Internal(x) \wedge String(y)$

$\forall x, y.\ night\_scene(x, y) \supset External(x) \wedge Boolean(y)$

$\forall x, y.\ name(x, y) \supset Location(x) \wedge String(y)$
$\forall x, y.\ address(x, y) \supset Location(x) \wedge String(y)$
$\forall x, y.\ description(x, y) \supset Location(x) \wedge Text(y)$

$\forall x.\ Scene(x) \supset (1 \leq \sharp\{y \mid code_{Scene}(x, y)\} \leq 1)$
. . .

$\forall x, y.\ stp\_for\_scn(x, y) \supset Setup(x) \wedge Scene(y)$
$\forall x, y.\ tk\_of\_stp(x, y) \supset Take(x) \wedge Setup(y)$
$\forall x, y.\ located(x, y) \supset External(x) \wedge Location(y)$

$\forall x.\ Setup(x) \supset 1 \leq \sharp\{y \mid stp\_for\_scn(x, y)\} \leq 1$
$\forall y.\ Scene(y) \supset 1 \leq \sharp\{x \mid stp\_for\_scn(x, y)\}$
$\forall x.\ Take(x) \supset 1 \leq \sharp\{y \mid tk\_of\_stp(x, y)\} \leq 1$
$\forall x.\ Setup(y) \supset 1 \leq \sharp\{x \mid tk\_of\_stp(x, y)\}$
$\forall x.\ External(x) \supset 1 \leq \sharp\{y \mid located(x, y)\} \leq 1$

$\forall x.\ Internal(x) \supset Scene(x)$
$\forall x.\ External(x) \supset Scene(x)$
$\forall x.\ Internal(x) \supset \neg External(x)$
$\forall x.\ Scene(x) \supset Internal(x) \vee External(x)$

On the contrary : hard for humans, difficult to generate

- Assign formal semantics to constructs of the conceptual design diagram
- Use it as usual
- Read diagrams as logical theories

LOOK AT SLIDES FOR UML RECAP

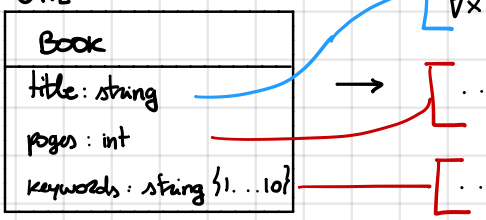
ATTRIBUTES formalization

FOL

$\forall x,y.\ title(x,y) \supset Book(x) \wedge String(y)$

$\forall x.\ Book(x) \supset (1 \leq \#\{y \mid title(x,y)\} \leq 1)$ (→ only one title)

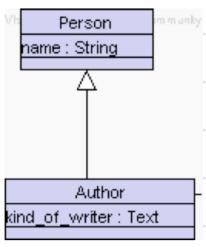
UML



Each attributes is described using two FOL formulas
Second one (multiplicity) is a short hand for:
- At least 1: $\exists y, a(x,y)$
- At most
  - 1: $\forall y_1, y_2, a(x,y_1) \wedge a(x,y_2) \supset y_1 = y_2$
  - x: $\exists y_1 \dots y_m \quad a(x,y_1) \wedge \dots \wedge a(x,y_m) \wedge y_1 \neq y_2 \wedge y_1 \neq y_2 \wedge \dots \wedge y_{m-1} \neq y_m$

Some thing can be done with ASSOCIATIONS
eg $\forall x,y$ written_by $(x,y) \supset Book(x) \wedge Author(y)$
$\forall x$ $Book(x) \supset (1 \leq \#\{y \mid written\_by(x,y)\})$

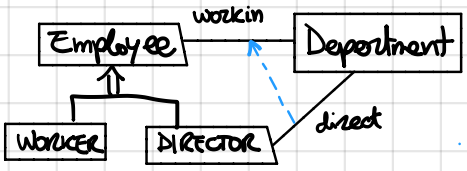
ISA



$\forall x$ $Author(x) \supset Person(x)$
$\forall x$ $name(x,y) \supset Person(x) \wedge String(y)$

$\forall x$ $Author(x) \supset 1 \leq \#\{y \mid kow(x,y)\} \leq 1$
$\forall xy$ $kow(x,y) \supset Author(x) \wedge Text(y)$

This works also for "subsets" of associations



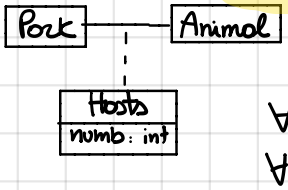
$\forall x,y$ $Directs(x,y) \supset worksin(x,y)$

Note: direct inherits max multiplicity from workin
workin inherits min multiplicity from direct
This is logically implied anyway.

complete: $\forall x$ $E(x) \supset Dir(x) \vee W(x)$
disjoint: $\forall x$ $W(x) \supset \neg Dir(x)$ Note: viceversa not needed

Association Class = association with attributes

eg

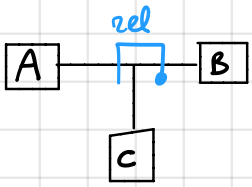


$\forall xyz$ $numbers(x,y,z) \supset hosts(x,y) \wedge int(z)$
$\forall xy$ $hosts(x,y) \supset 1 \leq \#\{z \mid numbers(x,y,z)\} \leq 1$
+ normal stuff of classic associations

KEY

$\forall x y \; Pd(x,y) \supset P(x) \wedge D(y)$
$\forall x y \; Pp(x,y) \supset P(x) \wedge P_2(y)$
+ multiplicity

$\forall x x' y z \quad Pd(x,y) \wedge Pd(x',y) \wedge Pp(x,z) \wedge Pp(x',z) \supset x = x'$



$\forall x y z \; rel(x,y,z) \supset A(x) \wedge B(y) \wedge C(z)$ as always
plus $\forall x y z z' \; rel(x,y,z) \wedge rel(x,y,z') \supset z = z'$

FORMS OF REASONING

Let be Γ the set of FOL assertions corresponding to UML diagram and $C(x)$ the predicate of class C.

C is consistent iff: $\Gamma \not\models \forall x \; C(x) \supset false$ i.e there must exist a model of Γ where the extension of $C(x)$ is not the empty set

The diagram is consistent iff Γ is satisfiable i.e Γ admits at least one model

C_1 subsumes C_2 if the class diagram implies that C_1 is a generalization of C_2 : $\Gamma \models \forall x \; C_2 \supset C_1(x)$

Two classes are equivalent if they denote the same set of instances whenever the conditions imposed by diagram are satisfied

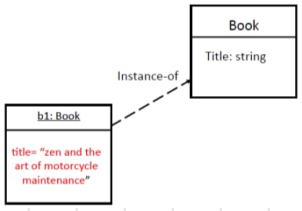
CHECKING UML DIAGRAMS INSTANTIATIOS

An instatiation (object diagram) is an extension to all classes, associations and atributes, describing properties of single objects or relations between them ⟶ Describe actual data = database.

Two cases:
① NO ISA : diagram acts as constraints
② ISA : instantiation is partial, not explicit instantiate superclasses. Start from "complete" form.

① In this case the instantiation of the diagram is a first order interpretation of FOL :
$I = (Obj^I, C_1^I \dots, C_n^I, A_1^I \dots, A_m^I, T_1^I \dots T_e^I, a^I \dots a_k^I$

eg.



$B(x), t(x,y), String(x)$

UML Class Diagram Axioms

$\forall x, y.t(x,y) \rightarrow B(x) \wedge String(y)$
$\forall x.B(x) \rightarrow 1 \leq \sharp\{y \mid t(x,y)\} \leq 1$

Γ

⟶

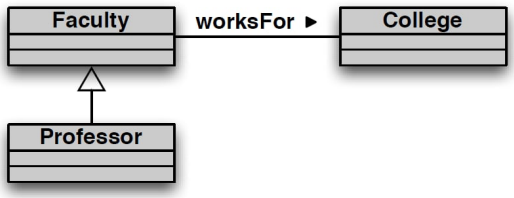
Instantiation

$\mathcal{I} = (Obj^{\mathcal{I}}, B^{\mathcal{I}}, String^{\mathcal{I}}, t^{\mathcal{I}})$

$B^{\mathcal{I}} = \{b1\}$
$t^{\mathcal{I}} = \{(b1, "zen")\}$
$String^{\mathcal{I}} = all\ strings$
$Obj^{\mathcal{I}} = \{b1\}$

If the instantiation is correct we can query it Just computing query q over $\mathcal{I}$ ignoring constraints Γ

② ISA - complete : each object in the subclass is also in the superclass



$\mathcal{I}$: $\text{Faculty}^{\mathcal{I}} = \{\text{john}, \text{mary}, \text{paul}\}$
$\text{Professor}^{\mathcal{I}} = \{\text{john}, \text{paul}\}$
$\text{College}^{\mathcal{I}} = \{\text{collA}, \text{collB}\}$
$\text{worksFor}^{\mathcal{I}} = \{(\text{john},\text{collA}),(\text{mary},\text{collB})\}$

ISA - NOT complete : for every object it is assumed that the instantiation state explicitly which are the most specific classes it is instance of

$\mathcal{I}$: $\text{Faculty}^{\mathcal{I}} : \{\text{mary}\}$ – *incomplete!*
$\text{Professor}^{\mathcal{I}} : \{\text{john}, \text{paul}\}$
$\text{College}^{\mathcal{I}} : \{\text{collA}, \text{collB}\}$
$\text{worksFor}^{\mathcal{I}} : \{(\text{john},\text{collA}),(\text{mary},\text{collB})\}$

In this case, when answering query, we have to use complete instantiation. eg. John and Paul are still "Faculty" too

STRUCTURAL OPERATIONAL SEMANTICS OF PROGRAM

Evolution Semantic: $(\delta, s) \longrightarrow s'$ where δ is a program, s is memory state before evaluation and s' the state after the evaluation.

How do we define this relation?

STRUCTURAL RULES

$$\frac{\text{CONSEQUENT}}{\text{ANTECEDENT}} \quad \text{IF} \quad \text{SIDE-CONDITION} \qquad \Rightarrow \forall(\text{ANT} \wedge \text{SIDE} \supset \text{CONS})$$

$$Act: \frac{(a, s) \longrightarrow s'}{true} \quad \text{if } s \models Pre(a) \text{ and } s' = Post(a, s)$$

special case: assignment $\frac{(x := v, s) \longrightarrow s'}{true}$ if $s' = s[x = v]$

$$Skip: \frac{(skip, s) \longrightarrow s}{true}$$

$$Seq: \frac{(\delta_1; \delta_2, s) \longrightarrow s'}{(\delta_1, s) \longrightarrow s'' \ \wedge \ (\delta_2, s'') \longrightarrow s'}$$

$$if: \frac{(\text{if } \phi \text{ then } \delta_1 \text{ else } \delta_2, s) \longrightarrow s'}{(\delta_1, s) \longrightarrow s'} \text{ if } s \models \phi \qquad \frac{(\text{if } \phi \text{ then } \delta_1 \text{ else } \delta_2, s) \longrightarrow s'}{(\delta_2, s) \longrightarrow s'} \text{ if } s \models \neg\phi$$

$$while: \frac{(\text{while } \phi \text{ do } \delta, s) \longrightarrow s}{true} \text{ if } s \models \neg\phi \qquad \frac{(\text{while } \phi \text{ do } \delta, s) \longrightarrow s'}{(\delta, s) \longrightarrow s'' \ \wedge \ (\text{while } \phi \text{ do } \delta, s'') \longrightarrow s'} \text{ if } s \models \phi$$

ex Compute S_f in following cases assuming that in S_0 we have $x=10$ and $y=0$

- $(X := x+1;\ x := x*2, S_0) \longrightarrow S_f$

↳ $\dfrac{(X := x+1;\ x := x*2, S_0) \longrightarrow S_f}{(X := x+1, S_0) \longrightarrow S_1 \ \wedge \ (x := x*2, S_1) \longrightarrow S_f}$ (true) (true)

S_1: $x = 11$, $y = 0$ $\qquad$ S_f: $x = 22$, $y = 0$

- $(x := x+1;\ \text{if } (x<10) \text{ then } x := 0 \text{ else } x := 1;\ x := x+1, S_0) \longrightarrow S_f$

↳

$$\frac{(x := x+1;\ \text{if } (x<10) \text{ then } x := 0 \text{ else } x := 1;\ x := x+1, S_0) \longrightarrow S_f}{x := x+1, S_0 \longrightarrow S_1 \ \wedge \ \dfrac{(\text{if } \ldots, x := x+1, S_1) \longrightarrow S_f}{\dfrac{(\text{if } \ldots, S_1) \longrightarrow S_2 \ \wedge \ (x+1, S_2) \longrightarrow S_f}{x := 1, S_1 \longrightarrow S_2}}}$$

(true) (true) (true)

S_1: $x = 11$, $y = 0$

S_2: $x = 1$, $y = 0$

S_f: $x = 12$, $y = 0$

TRANSITION SEMANTICS

This kind of semantic just compute a single step: $\delta, s \rightarrow \delta', s'$

Given δ and s, compute the state s' and the program δ' that remains to be executed obtained by executing a single step of δ in s

Note: long arrow $\longrightarrow$ for normal semantic, short arrow $\rightarrow$ for transition semantics.

ϵ is the empty program

$$Act: \frac{(a,s) \longrightarrow (\epsilon, s')}{true} \quad \text{if } s \models Pre(a) \text{ and } s' = Post(a,s)$$

special case: assignment $\frac{(x := v, s) \longrightarrow (\epsilon, s')}{true}$ if $s' = s[x = v]$

$$Skip: \frac{(skip, s) \longrightarrow (\epsilon, s)}{true}$$

$$Seq: \frac{(\delta_1; \delta_2, s) \longrightarrow (\delta_1'; \delta_2, s')}{(\delta_1, s) \longrightarrow (\delta_1', s')} \qquad \frac{(\delta_1; \delta_2, s) \longrightarrow (\delta_2', s')}{(\delta_2, s) \longrightarrow (\delta_2', s')} \text{ if } (\delta_1, s)\checkmark$$

↳ if δ_1 is terminated $= \epsilon$

$$if: \frac{(\textbf{if } \phi \textbf{ then } \delta_1 \textbf{ else } \delta_2, s) \longrightarrow (\delta_1', s')}{(\delta_1, s) \longrightarrow (\delta_1', s')} \text{ if } s \models \phi \qquad \frac{(\textbf{if } \phi \textbf{ then } \delta_1 \textbf{ else } \delta_2, s) \longrightarrow (\delta_2', s')}{(\delta_2, s) \longrightarrow (\delta_2', s')} \text{ if } s \models \neg\phi$$

$$while: \frac{(\textbf{while } \phi \textbf{ do } \delta, s) \longrightarrow (\delta'; \textbf{while } \phi \textbf{ do } \delta, s')}{(\delta, s) \longrightarrow (\delta', s')} \text{ if } s \models \phi$$

TERMINATION RULES

$$\epsilon: \frac{(\epsilon, s)\checkmark}{true}$$

$$Seq: \frac{(\delta_1; \delta_2, s)\checkmark}{(\delta_1, s)\checkmark \ \wedge \ (\delta_2; s)\checkmark}$$

$$if: \frac{(\textbf{if } \phi \textbf{ then } \delta_1 \textbf{ else } \delta_2, s)\checkmark}{(\delta_1, s)\checkmark} \text{ if } s \models \phi \qquad \frac{(\textbf{if } \phi \textbf{ then } \delta_1 \textbf{ else } \delta_2, s)\checkmark}{(\delta_2, s)\checkmark} \text{ if } s \models \neg\phi$$

$$while: \frac{(\textbf{while } \phi \textbf{ do } \delta, s)\checkmark}{true} \text{ if } s \models \neg\phi \qquad \frac{(\textbf{while } \phi \textbf{ do } \delta, s)\checkmark}{(\delta, s)\checkmark} \text{ if } s \models \phi$$

Eg Compute δ', s' assuming in S_0 $x=10$ and $y=0$

- $(\text{while } (y<2) \text{ do } \{x := x*2, y := y+1\}, S_0) \longrightarrow S_f$ long arrow

$S_2 = \{x=20, y=0\}$ $S_1 = \{x=20, y=1\}$
$S_4 = \{x=40, y=1\}$ $S_3 = \{x=40, y=2\}$

$(x := x*2; y := y+1; S_0) \longrightarrow S_1 \ \wedge \ (\text{while } \_, S_1 \longrightarrow S_f)$

$(x := x*2, S_0) \longrightarrow S_2 \ \wedge \ (y := y+1, S_2) \longrightarrow S_1$ ↳ $(x := x*2; y := y+1; S_1) \longrightarrow S_3 \wedge (\text{while } \_ S_3) \longrightarrow S$

true true

$(x := x*2, S_1) \longrightarrow S_4 \ \wedge \ (y := y+1; S_4) \longrightarrow S_3$

true true

Some thing but with transition semantics

$(\text{while } (y<2) \text{ do } \{x := x*2, y := y+1\}, S_0) \longrightarrow (\delta_1; \text{while } \_; S_1)$

$\delta_1 = \delta_2; y = y+1 = \epsilon; y = y+1$
$\delta_2 = \epsilon$
$S_1 = \{x=20, y=0\}$

$$\frac{(x = x*2; y = y+1, S_0) \longrightarrow (\delta_1, S)}{(x = x*2, S_0) \xrightarrow{\delta_2} (\epsilon, S_1)}$$

$$\frac{\epsilon; y := y+1; \text{while } \_, S_1 \longrightarrow \delta_3, \text{while } \_, S_2}{y := y+1; \text{while } \_, S_1 \longrightarrow \delta_3, S_2}$$

$$y = y+1, S_1 \longrightarrow (\epsilon, S_2)$$

$\delta_3 = \epsilon$, while
$S_2 = \{x=20, y=1\}$

Continue in my nightmares...

To characterize a whole computation using single steps: $\longrightarrow^*$

$$0 \text{ step}: \quad \frac{(\delta, s) \longrightarrow^* (\delta, s)}{true}$$

$$n \text{ step}: \quad \frac{(\delta, s) \longrightarrow^* (\delta'', s'')}{(\delta, s) \longrightarrow (\delta', s') \wedge (\delta', s') \longrightarrow^* (\delta'', s'')} \quad (\text{for some } \delta', s')$$

Notice that such relation is the reflexive-transitive closure of (single step) $\longrightarrow$.

Th. For every while-program δ and states s and s_f:
$(\delta, s_0) \longrightarrow s_f \equiv (\delta, s_0) \rightarrow^* (\delta_f, s_f) \wedge (\delta_f, s_f)^{\checkmark}$ for some δ_f

Why we do this transaction semantic? Because it can handle concurrency.

- $(\delta_1 \parallel \delta_2)$ concurrent execution
- if φ then δ_1 else δ_2 synchronized conditional
- while φ do δ synchronized loop

] test is "free", no transition used for it.

$$\text{transition}: \quad \frac{(\delta_1 \parallel \delta_2, s) \longrightarrow (\delta_1' \parallel \delta_2, s')}{(\delta_1, s) \longrightarrow (\delta_1', s')} \qquad \frac{(\delta_1 \parallel \delta_2, s) \longrightarrow (\delta_1 \parallel \delta_2', s')}{(\delta_2, s) \longrightarrow (\delta_2', s')}$$

$$\text{termination}: \quad \frac{(\delta_1 \parallel \delta_2, s)^{\checkmark}}{(\delta_1, s)^{\checkmark} \wedge (\delta_2, s)^{\checkmark}}$$

Presence of $\delta_1 \parallel \delta_2$ makes the transition relation NONDETERMINISTIC

HOARE LOGIC

Used to reason about the correctness of programs.
We start with some language we already saw (eg skip, assignment etc)

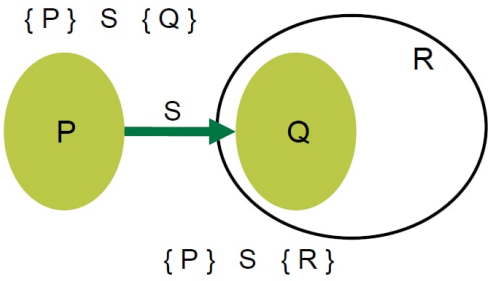
Given state $s \models P$ and $s' \models Q$, with program δ we want to check that
$s \models P \wedge \delta, s \longrightarrow s' \supset s' \models Q$.

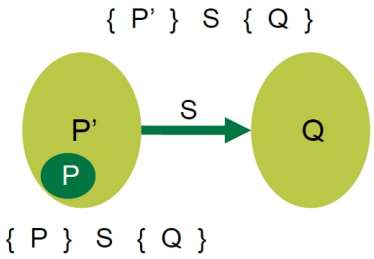
$\chi_P = \{ s \mid s \models P \}$ = set of all states satisfying P
$P(s) = s \in \chi_P$

P is stronger than Q, and is weaker than P if $P \Rightarrow Q$

$\{P\}$ Pr(ogram) $\{Q\} = \forall s \; s \models P \Rightarrow (\forall s' \; \underbrace{s \; Pr \; s'}_{\delta, s \longrightarrow s'} \Rightarrow s' \models Q)$

Since the model does not express non-termination, we assume Pr terminates.
With this assumption we talk about partial correctness, total otherwise.





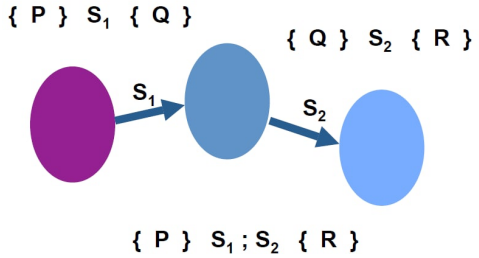
Post-condition weakening Rule:

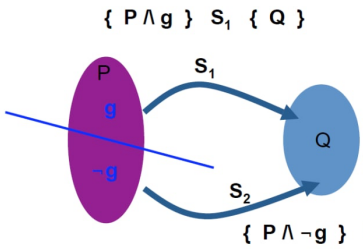
$$\frac{\{P\}\ S\ \{Q\} \quad , Q \Rightarrow R}{\{P\}\ S\ \{R\}}$$

Pre-condition strengthening Rule:

$$\frac{P \Rightarrow P' \ , \ \{P'\}\ S\ \{Q\}}{\{P\}\ S\ \{Q\}}$$

Premise 1, 2
↓
Conclusion





{ P } if g then S_1 else S_2 { Q }

$$\frac{\{P\}\ S_1\ \{Q\} \ , \ \{Q\}\ S_2\ \{R\}}{\{P\}\ S_1 ; S_2\ \{R\}}$$

$$\frac{\{P \wedge g\}\ S_1\ \{Q\} \ , \ \{P \wedge \neg g\}\ S_2\ \{Q\}}{\{P\}\ \text{if } g \text{ then } S_1 \text{ else } S_2\ \{Q\}}$$

WEAKEST PRECONDITION

wp: Statement × Pred ⟶ Pred

Let $W = wp(S, Q)$:

- Reachability: from any $s \models W$ if $s \llbracket S \rrbracket s'$ then $s' \models Q$
- Maximality: $s \llbracket S \rrbracket s'$ and $s' \models Q$ implies $s \models W$

$\Rightarrow \{P\}\ S\ \{Q\} = P \Rightarrow wp(S, Q)$

- $wp\ skip\ Q = Q$
- $wp\ (x := e)\ Q = Q[e/x]$
- $wp((S_1 ; S_2), Q) = wp(S_1, (wp(S_2, Q)))$
- $wp((\text{if } g \text{ then } S_1 \text{ else } S_2), Q) = g \wedge wp(S_1, Q) \vee \neg g \wedge wp(S_2, Q)$

Note: No while!

Ex. Compute the weakest precondition of getting $x = y$ from following program

```
S  x := y + 1
   if (y > 0) then
        x := x + y
   else x := y + 100
   x := x + y
```

$wp(S, x = y)$ ⟶

$\{(y > 0 \wedge y + 1 = -y) \vee (y = -100)\} = \{y = -100\}$ (y = -1/2, False)

$x := y + 1$

$\{(y > 0 \wedge x = -y) \vee (y \leq 0 \wedge y = -100)\}$

if $(y > 0)$ then

$\{x + y = 0\} = \{x = -y\}$

$x := x + y$

some: $\{x = 0\}$ $\{y + 100 = 0\} = \{y = -100\}$

else: $x := y + 100$ $\{x = 0\}$

$\{x + y = y\} = \{x = 0\}$

$x := x + y$

$\{x = y\}$

WHILE

$P \Rightarrow I$

$\{g \wedge I\}\ S\ \{I\}$ ← setting up invariance

$I \wedge \neg g \Rightarrow Q$ ← exit condition

$\{P\}$ while g do S $\{Q\}$

Ex Check if following Hoare triple is correct using as invariant $(0 \le i \wedge 0 \le J \wedge i+J \le 5)$

$\{i=0 \text{ AND } J=5\}$ while $(i<5)$ do $\underbrace{(J=J-1;\ i:=i+1)}_{\delta}$ $\{J=0\}$

Solution: $P = \{i=0 \text{ AND } J=5\}$

$Q = \{J=0\}$

$I = (0 \le i \wedge 0 \le J \wedge i+J \le 5)$

1. Check $P \supset I$

$i=0 \wedge J=5 \supset 0 \le i \wedge 0 \le J \wedge i+J \le 5$ ✓

3. Check $\neg g \wedge I \supset Q$

$i \ge 5 \wedge 0 \le i \wedge 0 \le J \wedge i+J \le 5 \supset J=0$ ✓

$\llcorner\ i=5 \wedge J=0$

2. Check $\{g \wedge I\}\ \delta\ \{I\} \quad = \quad g \wedge I \supset wp(\delta, I)$

$i<5 \wedge 0 \le i \wedge 0 \le J \wedge i+J \le 5 \supset wp(\delta, I)$

$\{0 \le i+1 \wedge 0 \le J-1 \wedge i+1+J-1 \le 5\}$

$J := J-1$

$\{0 \le i+1 \wedge 0 \le J \wedge i+1+J \le 5\}$

$i := i+1$

$\{0 \le i \wedge 0 \le J \wedge i+J \le 5\}$

checking i and J we can say that the Hoare triple is correct using I as invariant

Ex Compute weakest precondition for getting $\{x=0\}$ for following program:

```
x := 50+y
if (x>50) then {
    if (y>0) then {
        x := x-y
    else y := y
    }
else x := x+y
y := y+50
```

$\{y=-25\}$

$50=0 \qquad \{y \le 50 \wedge y=-25\}$

$\{50+y>50 \wedge y=50+y \vee 50+y \le 50 \wedge 50+y+y=0\}$

$x := 50+y$

$\{x>50 \wedge y=x \vee x \le 50 \wedge x+y=0\}$

$x>50 \wedge (y>0 \wedge x=y \vee y \le 0 \wedge x=0) \vee (x \le 50 \wedge x+y=0)$

$x>50$

if $(x>50)$ then $\{$

$y>0 \wedge x=y \vee y \le 0 \wedge x=0$

if $(y>0)$ then

$\{x-y=0\}$

$x := x-y$

$\{x=0\}$

else $y := y$

$\{x=0\}$

$\{x+y=0\}$

else $x := x+y$

$\{x=0\}$

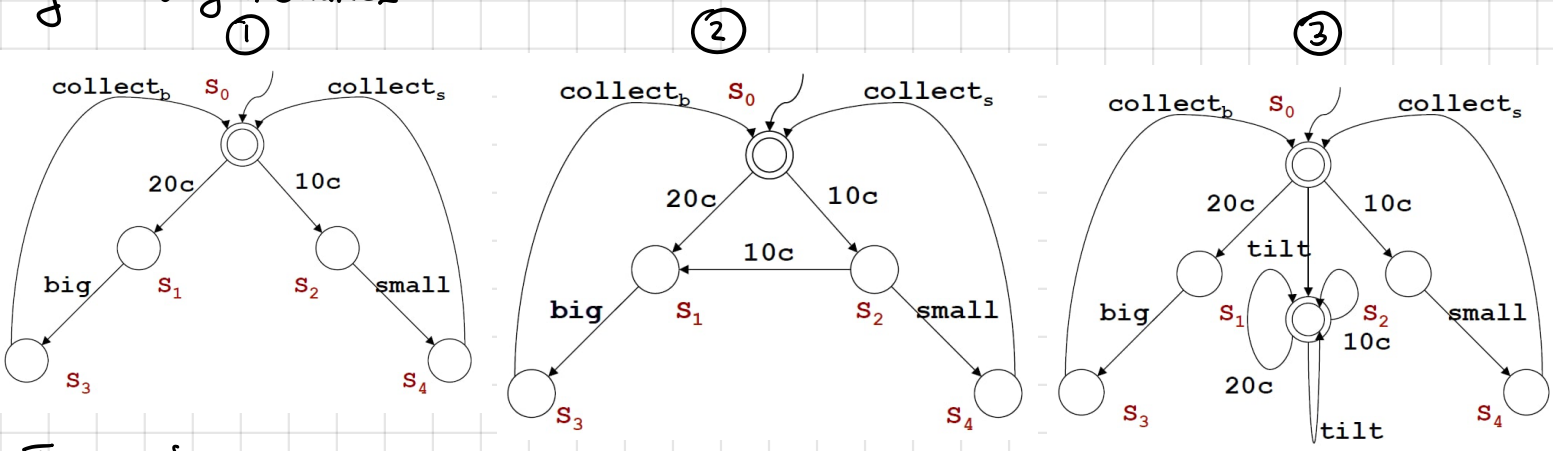
$y := y+50$

$\{x=0\}$

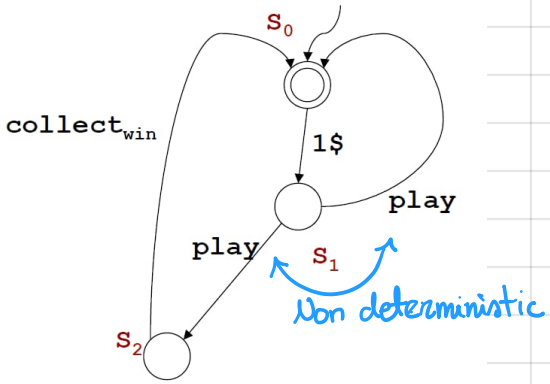
TRANSITION SYSTEMS AND BISIMULATION

Graph of transitions. Termination and reachability (of final state) let us query the system (fixpoint theory, model checking)

Eg vending machines



Eg Slot machine NON DETERMINISTIC



A transition system TS is a tuple $T = \langle A, S, S^0, \delta, F \rangle$ where:

- A = set of actions
- S = set of states
- $S^0 \subseteq S$ set of initial state
- $\delta \subseteq S \times A \times S$ = transition relation
- $F \subseteq S$ = set of final states

+ equivalent relations!

REACHABILITY

A binary relation R is a reachability like relation iff:

- $(s, s) \in R$
- if $\exists\, a, s'$ s.t. $s \rightarrow s_a \wedge (s', s'') \in R$ then $(s, s'') \in R$

A state S_0 reaches a state S_f iff $\forall$ reachability-like relations R we have $(S_0, S_f) \in R$

Algorithm ComputingReachability

Input: transition system TS
Output: the **reachable-from** relation (the smallest reachability-like relation)

Body

```
R = ∅
R' = {(s,s) | s ∈ S}
while (R ≠ R') {
    R := R'
    R' := R' ∪ {(s,s'') | ∃ s',a. s →a s' ∧ (s',s'') ∈ R }
}
return R'
```

YdoB

BISIMULATION

Two transition systems are bisimilar if they have the same behaviour:
- locally they look indistinguishable
- every action that can be done on one of them can also be done on the other remaining indistinguishable.

R is a bisimulation iff $(s,t) \in R$ implies that:
- s is final iff t is final
- $\forall$ actions a:
  - if $s \rightarrow_a s'$ then $\exists t'$ s.t. $t \rightarrow_a t'$ and $(s',t') \in R$
  - if $t \rightarrow_a t'$ then $\exists s'$ s.t. $s \rightarrow_a s'$ and $(s',t') \in R$

A state s_0 is bisimilar (or equivalent) to state t_0 iff there exists a bisimulation between the initial states s_0 and t_0

LEAST AND GREATEST FIXPOINTS

Consider $X = f(X)$ where f is an operator from 2^S to 2^S (2^S = set of all subsets of S)
Every solution E of this equation is called fixpoint of f. Smallest and greatest solution are called least and greatest fixpoints.

Operator from subset of status to subset of status

Every E s.t. $f(E) \subseteq E$ is called pre-fixpoint
Every E s.t. $E \subseteq f(E)$ is called post-fixpoint

f is monotonic if $E_1 \subseteq E_2$ implies $f(E_1) \subseteq f(E_2)$. In this case:
Th. • there exists a unique least fixpoints of f which is given by: $\cap \{E \subseteq S \mid f(E) \subseteq E\}$ (pre-fixpoints)
• there exists a unique greatest fixpoint of f which is given by: $\cup \{E \subseteq S \mid E \subseteq f(E)\}$ (post-fixpoints)

APPROXIMATES

The approximates for a least fixpoint $\mathcal{L} = \cap \{E \subseteq S \mid f(E) \subseteq E\}$ are as follow:

$$\left.\begin{array}{l} Z_0 = \emptyset \\ Z_1 = f(Z_0) \\ Z_2 = f(Z_1) \\ \dots \end{array}\right\} \quad \forall i,\ Z_i \subseteq Z_{i+1} \quad , \quad \forall i\ Z_i \subseteq \mathcal{L}$$

Th. If for some n $Z_{n+1} = Z_n$ then $Z_n = \mathcal{L}$

Least fixpoint algorithm

```
Z_old := ∅;
Z := f(Z_old);
while (Z ≠ Z_old){
    Z_old := Z;
    Z := f(Z);
}
```

Similar for greatest fixpoint $\mathcal{G} = \cup \{E \subseteq S \mid E \subseteq f(E)\}$.

$$\left.\begin{array}{l} Z_0 = S \\ Z_1 = f(Z_0) \\ Z_2 = f(Z_1) \\ \dots \end{array}\right\} \quad \forall i\ Z_{i+1} \subseteq Z_i \quad , \quad \forall i\ \mathcal{G} \subseteq Z_i$$

Contrary.

Th. If for some n, $Z_{n+1} = Z_n$ then $Z_n = \mathcal{G}$

Greatest fixpoint algorithm

```
Z_old := S;
Z := f(Z_old);
while (Z ≠ Z_old){
    Z_old := Z;
    Z := f(Z);
}
```

μ - CALCULUS

Logic based on fixpoints. Composed by:
- Propositions: to denote properties of the global store in a given configuration
- Modalities: to denote the capability of performing certain actions in a given configuration
- Least/Greatest Fixpoints: to denote "temporal" properties of the system (induction / coinduction)

μ-calculus syntax

modal operator — least — greatest

$$\Phi ::= A \mid true \mid false \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \Phi_1 \vee \Phi_2 \mid \langle a\rangle\Phi \mid [a]\Phi \mid \mu X.\Phi \mid \nu X.\Phi \mid X$$

FOL — diamond — box — fixpoint theory

Diamond = exist a state where Φ holds
Box = every states imply Φ

For formulae of the form $\mu X.\Phi$ and $\nu X.\Phi$ we require syntactic monotonicity of Φ.
Every occurrence of variable X in Φ must be within the scope of an even number of negation signs.
In μ-calculus, given the requirement of syntactic monotonicity, the least fixpoint and the greatest one ALWAYS exist.

Transition System: $T = (S, \{R_a \mid a \in A\}, \Pi)$
P = given proposition set → S = set of states
A = given set of atomic actions → R_a = family of relations
Π = mapping from P to a subset of S

Valuation: given transition system T, a valuation V on T is a mapping from variables in Var to subsets of states in T.

μ-calculus semantics

$$\begin{aligned}
(A)^T_V &= \Pi(A) \subseteq S \\
(X)^T_V &= V(X) \subseteq S \\
(true)^T_V &= S \\
(false)^T_V &= \emptyset \\
(\neg\Phi)^T_V &= S - (\Phi)^T_V \\
(\Phi_1 \wedge \Phi_2)^T_V &= (\Phi_1)^T_V \cap (\Phi_2)^T_V \\
(\Phi_1 \vee \Phi_2)^T_V &= (\Phi_1)^T_V \cup (\Phi_2)^T_V \\
(\langle a\rangle\Phi)^T_V &= \{s \in S \mid \exists s'.\ (s,s') \in R_a \text{ and } s' \in (\Phi)^T_V\} \\
([a]\Phi)^T_V &= \{s \in S \mid \forall s'.\ (s,s') \in R_a \text{ implies } s' \in (\Phi)^T_V\} \\
(\mu X.\Phi)^T_V &= \bigcap\{\mathcal{E} \subseteq S \mid (\Phi)^T_{V[X\leftarrow\mathcal{E}]} \subseteq \mathcal{E}\} \quad \text{least fp. of } \Phi \\
(\nu X.\Phi)^T_V &= \bigcup\{\mathcal{E} \subseteq S \mid \mathcal{E} \subseteq (\Phi)^T_{V[X\leftarrow\mathcal{E}]}\} \quad \text{greatest fp. of } \Phi
\end{aligned}$$

Given $T = (S, \{R_a \mid a \in 2^A\}, \Pi)$
- $(\Phi)^T_V$ = set of states that satisfy Φ

Examples:
1) ⟨next⟩true = capability of making a next-transition
2) [next]false = inability of making any next-transition
3) ⟨next⟩true ∧ [next]P = next-transition are allowed and all reach states where P holds
4) μX. P ∨ ⟨next⟩X = exists an evolution of the system s.t. P eventually holds.
5) νX.P ∧ [next]X = ¬μX.¬P∨⟨next⟩X = invariance of P under all of the evolutions of the system.
6) μX.P ∨(⟨next⟩true ∧ [next]X) = for all evolutions of the system P eventually holds
7) νX.μY.(P∧⟨next⟩X) ∨ (⟨next⟩Y) = strong fairness, there exists a run where P is true infinitely often

We are interested in model checking: querying the transition system.
Let $T = (S, \{R_a \mid a \in A\}, \Pi)$ be a transition system, let $s \in S$ be one of its states and Φ be a closed (no free variables) μ-calculus formula. The related model checking problem is to verify whether $s \in (\Phi)^T_v$ where v is any valuation since Φ is closed. Also written $T, s \models \Phi$ or just $s \models \Phi$

μ-calculus model checking algorithm

$$
\begin{array}{rcl}
[\![A]\!]^T_v & = & \Pi(A) \\
[\![X]\!]^T_v & = & v(X) \\
[\![true]\!]^T_v & = & S \\
[\![false]\!]^T_v & = & \emptyset \\
[\![\neg\Phi]\!]^T_v & = & S - [\![\Phi]\!]^T_v \\
[\![\Phi_1 \wedge \Phi_2]\!]^T_v & = & [\![\Phi_1]\!]^T_v \cap [\![\Phi_2]\!]^T_v \\
[\![\Phi_1 \vee \Phi_2]\!]^T_v & = & [\![\Phi_1]\!]^T_v \cup [\![\Phi_2]\!]^T_v
\end{array}
\qquad
\begin{array}{rcl}
[\![\langle a \rangle \Phi]\!]^T_v & = & \mathrm{PREE}(a, [\![\Phi]\!]^T_v) \\
[\![[a]\Phi]\!]^T_v & = & \mathrm{PREA}(a, [\![\Phi]\!]^T_v)
\end{array}
$$

$$
[\![\mu X.\Phi]\!]^T_v = \mathrm{LFP}X.[\![\Phi]\!]^T_v
\qquad
[\![\nu X.\Phi]\!]^T_v = \mathrm{GFP}X.[\![\Phi]\!]^T_v
$$

PreE$(a, \mathcal{E})$ = existential a-preimage of $\mathcal{E}$ $= \{s \in S \mid \exists s'.\ (s, s') \in R_a$ and $s' \in \mathcal{E}\}$
PreA$(a, \mathcal{E})$ = universal a-preimage of $\mathcal{E}$ $= \{s \in S \mid \forall s'.\ (s, s') \in R_a$ implies $s' \in \mathcal{E}\}$

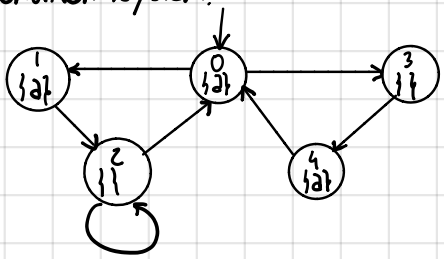
Procedure $\mathrm{LFP}X.[\![\Phi]\!]^T_v$

$\mathcal{X}_{old} := [\![False]\!]^T_v$;
$\mathcal{X} := [\![\Phi]\!]^T_{v[X \leftarrow \mathcal{X}_{old}]}$;
while $(\mathcal{X} \neq \mathcal{X}_{old})$ {
  $\mathcal{X}_{old} := \mathcal{X}$;
  $\mathcal{X} := [\![\Phi]\!]^T_{v[X \leftarrow \mathcal{X}_{old}]}$;
}
return $\mathcal{X}$;

Procedure $\mathrm{GFP}X.[\![\Phi]\!]^T_v$

$\mathcal{X}_{old} := [\![True]\!]^T_v$;
$\mathcal{X} := [\![\Phi]\!]^T_{v[X \leftarrow \mathcal{X}_{old}]}$;
while $(\mathcal{X} \neq \mathcal{X}_{old})$ {
  $\mathcal{X}_{old} := \mathcal{X}$;
  $\mathcal{X} := [\![\Phi]\!]^T_{v[X \leftarrow \mathcal{X}_{old}]}$;
}
return $\mathcal{X}$;

Ex Model check the μ-calculus formula $\mu X.\ \neg a \vee \langle next \rangle X$ against following transition system.



Lets define X_i the i-th approximation of X
$[X_i]$ = states that make formula true
$[X_0] = [false] = \emptyset$
$[X_1] = [\neg a \vee \langle next \rangle X_0] = [\neg a] \cup [\langle next \rangle X_0]$
$= [\neg a] \cup \mathrm{PreE}(\langle next \rangle, [X_0])$ ($[X_0] = \emptyset$)
$= \{2,3\} \cup \emptyset = \{2,3\}$

This set MUST get bigger!

$[X_2] = [\neg a \vee \langle next \rangle X_1] = [\neg a] \cup \mathrm{PreE}(next, [X_1]) = \{2,3\} \cup \{0,1,2\} = \{0,1,2,3\}$ Again! ok
$[X_3] = [\neg a \vee \langle next \rangle X_2] = [\neg a] \cup \mathrm{PreE}(next, [X_2]) = \{2,3\} \cup \{0,1,2,4\} = \{0,1,2,3,4\}$ Equal!
$[X_4] = [\neg a \vee \langle next \rangle X_3] = [\neg a] \cup \mathrm{PreE}(next, [X_3]) = \{2,3\} \cup \{0,1,2,3,4\} = \{0,1,2,3,4\}$ Least Fix Point

Since our initial state is in (the "solution"), we can say that is true that from initial state we can reach a state where $\neg a$ is true.

What if we change formula to $\nu X.\ \neg a \vee \langle next \rangle X$?
$[X_0]$ = entire set $= \{0,1,2,3,4\}$
$[X_1] = [\neg a \vee \langle next \rangle X] = [\neg a] \cup \mathrm{PreE}(next, [X_0]) = \{2,3\} \cup \{0,1,2,3,4\} = \{0,1,2,3,4\}$ Greatest Fix Point

$[\nu X.\ \neg a \wedge \langle next \rangle X] =$ $\neg a$ always true

$[X_0] = \{1,2,3,4\}$

In this case the set MUST shrink!

$[X_1] = [\neg a \wedge \langle next \rangle X_0] = [\neg a] \cap Pre\exists(next, [X_0]) = \{2,3\} \cap \{0,2,3,4\} = \{2,3\}$

$[X_2] = [\neg a \wedge \langle next \rangle X_1] = [\neg a] \cap Pre\exists(next, [X_1]) = \{2,3\} \cap \{0,1,2\} = \{2\}$

$[X_3] = [\neg a \wedge \langle next \rangle X_2] = [\neg a] \cap Pre\exists(next, [X_2]) = \{2,3\} \cap \{1,2\} = \{2\}$ Equal! Greatest FixPoint

Since our initial state is not in the fixpoint the formula is false

$[\mu X.\ \neg a \vee [next] X] =$ all executions reach $\neg a$ eventually

$[X_0] = \emptyset$

$[X_1] = [\neg a \vee [next] X_0] = [\neg a] \cup Pre\forall(next, [X_0]) = \{2,3\} \cup \{\emptyset\} = \{2,3\}$

$[X_2] = [\neg a \vee [next] X_1] = [\neg a] \cup Pre\forall(next, [X_1]) = \{2,3\} \cup \{1\} = \{1,2,3\}$

$[X_3] = [\neg a \vee [next] X_2] = [\neg a] \cup Pre\forall(next, [X_2]) = \{2,3\} \cup \{0,1\} = \{0,1,2,3\}$

$[X_4] = [\neg a \vee [next] X_3] = [\neg a] \cup Pre\forall(next, [X_3]) = \{2,3\} \cup \{0,1,2,4\} = \{0,1,2,3,4\}$

$[X_5] = [\neg a \vee [next] X_4] = [\neg a] \cup Pre\forall(next, [X_4]) = \{2,3\} \cup \{0,1,2,3,4\} = \{0,1,2,3,4\}$ Least Fix Point

Initial state in solution so formula is true

$[\nu X\ \mu Y.\ (a \wedge \langle next \rangle X \vee [next] Y)]$

$[X_0] = \{0,1,2,3,4\}$

$[X_1] = [\mu Y (a \wedge \langle next \rangle X_0) \vee [next] Y] = \{0,1,3,4\}$

$[Y_{10}] = \emptyset$

$[Y_{11}] = [a \wedge \langle next \rangle X_0 \vee [next] Y_{10}] = [a] \cap Pre\exists(next, [X_0]) \cup Pre\forall[next, Y_{10}] = \{0,1,4\} \cap \{0,1,2,3,4\} \cup \emptyset = \{0,1,4\}$

$[Y_{12}] = [a \wedge \langle next \rangle X_0 \vee [next] Y_{11}] = [a] \cap Pre\exists(next, [X_0]) \cup Pre\forall[next, Y_{11}] = \{0,1,4\} \cup \{4,3\} = \{0,1,3,4\}$

$[Y_{13}] = [a \wedge \langle next \rangle X_0 \vee [next] Y_{12}] = [\ldots] = \{0,1,4\} \cup \{0,3,4\} = \{0,1,3,4\}$ Least Fix Point "of X_1"

$[X_2] = [\mu Y (a \wedge \langle next \rangle X_1 \vee [next] Y] = \{0,3,4\}$

$[Y_{20}] = \emptyset$

$[Y_{21}] = [a \wedge \langle next \rangle X_1 \vee [next] Y_{20}] = \{0,1,4\} \cap \{0,2,3,4\} \cup \emptyset = \{0,4\}$

$[Y_{22}] = [a \wedge \langle next \rangle X_1 \vee [next] Y_{21}] = \{0,4\} \cup \{3,4\} = \{0,3,4\}$

$[Y_{23}] = [a \wedge \langle next \rangle X_1 \vee [next] Y_{22}] = \{0,4\} \cup \{3,4\} = \{0,3,4\}$ Least Fix Point "of X_2"

$[X_3] = [\mu Y (a \wedge \langle next \rangle X_2 \vee [next] Y] = \{0,3,4\}$

$[Y_{30}] = \emptyset$

$[Y_{31}] = [a \wedge \langle next \rangle X_2 \vee [next] Y_{30}] = \{0,1,4\} \cap \{0,2,3,4\} \cup \emptyset = \{0,4\}$

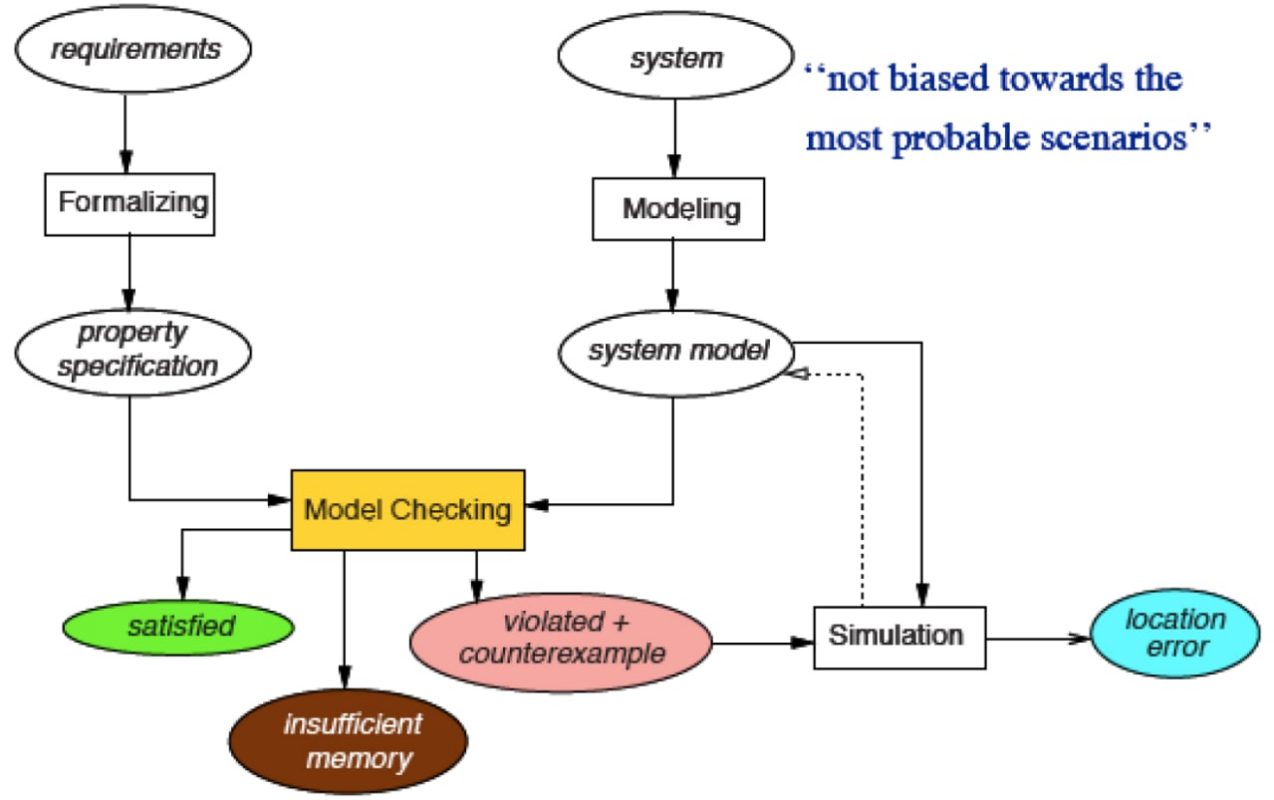
$[Y_{32}] = [a \wedge \langle next \rangle X_2 \vee [next] Y_{31}] = \{0,4\} \cup \{3,4\} = \{0,3,4\}$

$[Y_{33}] = [a \wedge \langle next \rangle X_2 \vee [next] Y_{32}] = \{0,4\} \cup \{3,4\} = \{0,3,4\}$ Least Fix Point "of X_3"

$X_2 = X_3 \rightarrow$ Greatest Fix Point

Initial state in solution $\rightarrow$ formula satisfied!

System Verification: check whether a system fulfills the qualitative requirements that have been identified. Checking we are building something in the right way.



LINEAR TEMPORAL LOGIC (LTL)

Only one timeline

$\bigcirc\varphi$	φ is true in the *next* moment in time
$\Box\varphi$	φ is true in *all* future moments
$\Diamond\varphi$	φ is true in *some* future moment
$\varphi \,\mathcal{U}\, \psi$	φ is true *until* ψ is true

SEMANTIC

$\langle M, i \rangle \models p$ iff $p \in I(i)$ for $p \in \Sigma$ = set of atomic propositions

$$\langle \mathcal{M}, i \rangle \models \neg\varphi \quad \textbf{iff} \quad \langle \mathcal{M}, i \rangle \not\models \varphi$$

$$\langle \mathcal{M}, i \rangle \models \varphi \wedge \psi \quad \textbf{iff} \quad \langle \mathcal{M}, i \rangle \models \varphi \text{ and } \langle \mathcal{M}, i \rangle \models \psi$$

$$\langle \mathcal{M}, i \rangle \models \varphi \vee \psi \quad \textbf{iff} \quad \langle \mathcal{M}, i \rangle \models \varphi \text{ or } \langle \mathcal{M}, i \rangle \models \psi$$

$$\langle \mathcal{M}, i \rangle \models \varphi \Rightarrow \psi \quad \textbf{iff} \quad \text{if } \langle \mathcal{M}, i \rangle \models \varphi \text{ then } \langle \mathcal{M}, i \rangle \models \psi$$

NEXT

Provides a constraint on the next moment in time. $\langle M, i \rangle \models \bigcirc\varphi$ iff $\langle M, i+1 \rangle \models \varphi$

eg $(sad \wedge \neg rich) \Rightarrow \bigcirc sad$

EVENTUALLY / SOMETIME

φ will be true but we dont know when. $\langle M, i \rangle \models \Diamond\varphi$ iff $\exists\, j\ (j \geq i) \wedge \langle M, j \rangle \models \varphi$

eg. $(\neg resigned \wedge sad) \Rightarrow \Diamond famous$

ALWAYS

Invariant properties. $\langle M, i \rangle \models \Box\varphi$ iff $\forall\, j$ if $(j \geq i)$ then $\langle M, j \rangle \models \varphi$

eg lottery-win $\Rightarrow \Box$ rich

UNTIL

$\langle M, i \rangle \models \varphi \,\mathcal{U}\, \psi$ iff $\exists\, j\ (j \geq i) \wedge \langle M, j \rangle \models \psi \wedge \forall\, k\ \ (i \leq k \leq j) \Rightarrow \langle M, k \rangle \models \varphi$

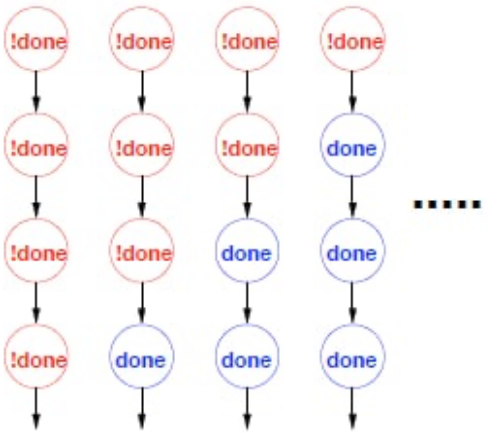
eg start lecture $\Rightarrow$ talk $\mathcal{U}$ end-lecture

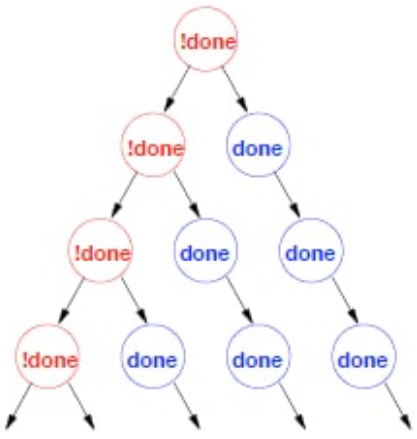
born $\Rightarrow$ alive $\mathcal{U}$ dead (φ cant lost forever and ψ sooner or later will happen)

Note:

1) $\neg\Box\varphi \equiv \Diamond\neg\varphi$
2) $\Diamond\varphi \equiv \top \,\mathcal{U}\, \varphi$ ($\top$ = true)

computation tree





Sometimes symbols are translated with letters:

◇ → Future

□ → Globally in the future

O → neXtime

COMPUTATION TREE LOGIC (CTL)

Based on the possibility of several "branches" (as we saw before).

CTL explicitly introduces path quantifiers:
- all paths = A
- exists a path = E

Use some temporal operators as LTL that can be combined with quantifiers:

Universal modalities AF, AG, AX, AU

Existential modalities EF, EG, EX, EU

Syntax for single state:

$\mathcal{KM}, s_i \models \neg\varphi$ **iff** $\mathcal{KM}, s_i \not\models \varphi$

$\mathcal{KM}, s_i \models \varphi \wedge \psi$ **iff** $\mathcal{KM}, s_i \models \varphi$ and $\mathcal{KM}, s_i \models \psi$

$\mathcal{KM}, s_i \models \varphi \vee \psi$ **iff** $\mathcal{KM}, s_i \models \varphi$ or $\mathcal{KM}, s_i \models \psi$

$\mathcal{KM}, s_i \models \varphi \Rightarrow \psi$ **iff** if $\mathcal{KM}, s_i \models \varphi$ then $\mathcal{KM}, s_i \models \psi$

$\mathcal{KM}, s_i \models \top$

$\mathcal{KM}, s_i \not\models \bot$

Same as before.

For a path $\pi = (s_i, s_{i+1}, \ldots)$ outgoing from state s_i:

$\mathcal{KM}, s_i \models \mathbf{AX}\varphi$ iff $\forall \pi = (s_i, s_{i+1}, \ldots)$ $\mathcal{KM}, s_{i+1} \models \varphi$

$\mathcal{KM}, s_i \models \mathbf{EX}\varphi$ iff $\exists \pi = (s_i, s_{i+1}, \ldots)$ $\mathcal{KM}, s_{i+1} \models \varphi$

$\mathcal{KM}, s_i \models \mathbf{AG}\varphi$ iff $\forall \pi = (s_i, s_{i+1}, \ldots)$ $\forall j \geq i. \mathcal{KM}, s_j \models \varphi$

$\mathcal{KM}, s_i \models \mathbf{EG}\varphi$ iff $\exists \pi = (s_i, s_{i+1}, \ldots)$ $\forall j \geq i. \mathcal{KM}, s_j \models \varphi$

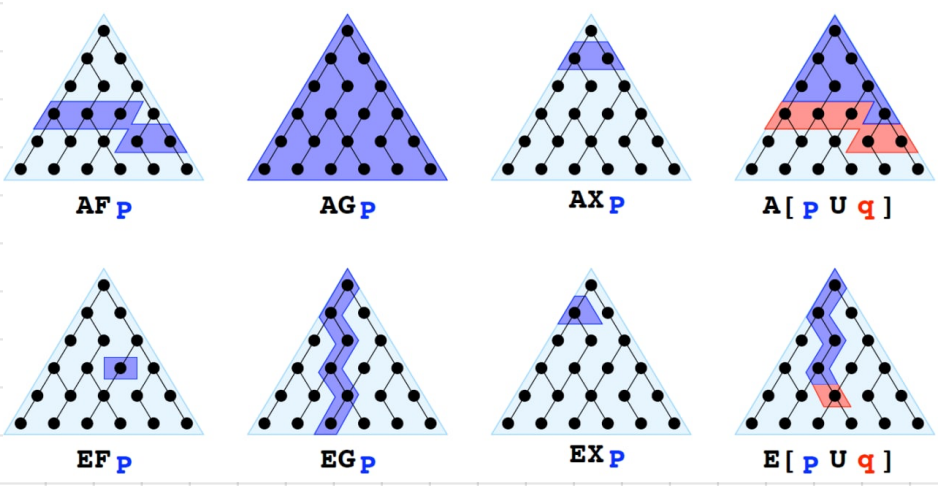
$\mathcal{KM}, s_i \models \mathbf{AF}\varphi$ iff $\forall \pi = (s_i, s_{i+1}, \ldots)$ $\exists j \geq i. \mathcal{KM}, s_j \models \varphi$

$\mathcal{KM}, s_i \models \mathbf{EF}\varphi$ iff $\exists \pi = (s_i, s_{i+1}, \ldots)$ $\exists j \geq i. \mathcal{KM}, s_j \models \varphi$

$\mathcal{KM}, s_i \models (\varphi \mathbf{AU} \psi)$ iff $\forall \pi = (s_i, s_{i+1}, \ldots)$ $\exists j \geq i. \mathcal{KM}, s_j \models \psi$ and $\forall i \leq k < j : M, s_k \models \varphi$

$\mathcal{KM}, s_i \models (\varphi \mathbf{EU} \psi)$ iff $\exists \pi = (s_i, s_{i+1}, \ldots)$ $\exists j \geq i. \mathcal{KM}, s_j \models \psi$ and $\forall i \leq k < j : \mathcal{KM}, s_k \models \varphi$

finally P | globally P | next P | P until q



Logic of CTL is based on combining these 8 operators!

MODEL CHECKING

Construct denotation of φ = set of states where formula holds and then compare it with set of initial states. Denotation of φ = $[\varphi]$

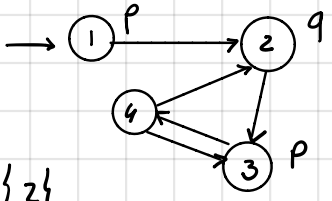
$EX p \leftrightarrow \langle next \rangle p$
$AX p \leftrightarrow [next]\, p$
} ⟶ we can use μ-calculus, we must "translate" to μ-calculus.

CTL	μ-calculus
p	p
$\vee, \wedge, \rightarrow$	$\vee, \wedge, \rightarrow$
$EX\varphi$	$\langle next \rangle t(\varphi)$
$AX\varphi$	$[next]\, t(\varphi)$
$EF\varphi$	$\mu Z\ t(\varphi) \vee \langle next \rangle Z$
$AF\varphi$	$\mu Z\ t(\varphi) \vee [next]\, Z$
$EG\varphi$	$\nu Z\ t(\varphi) \wedge \langle next \rangle Z$
$AG\varphi$	$\nu Z\ t(\varphi) \wedge [next]\, Z$
$\varphi E U \psi$	$\mu Z.\ t(\psi) \wedge (t(\varphi) \wedge \langle next \rangle Z)$
$\varphi A U \psi$	$\mu Z\ t(\psi) \vee (t(\varphi) \wedge [next]\, Z)$

Eg Model checking

$AG(p \Rightarrow AFq)$

$\alpha \quad \beta, \gamma$



$\alpha = AFq \rightarrow \mu Z\ q \vee [next]\, Z = \{1,2\}$

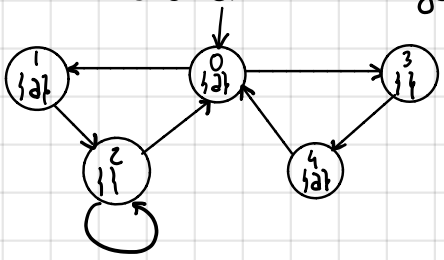
$[Z_0] = \emptyset$
$[Z_1] = [q \vee [next]\, Z_0] = [q] \cup Pre_A(next, [Z_0]) = \{2\}$
$[Z_2] = [q \vee [next]\, Z_1] = [q] \cup Pre_A(next, [Z_1]) = \{2\} \cup \{1\} = \{1,2\}$
$[Z_3] = [q \vee [next]\, Z_2] = [q] \cup Pre_A(next, [Z_2]) = \{2\} \cup \{1\} = \{1,2\}$

$\beta = \neg p \vee \alpha \rightarrow [\neg p] \cup [\alpha] = \{1,2,4\}$
$\gamma = AG\beta \rightarrow \nu Z\ \beta \wedge [next]\, Z = \emptyset$

$[Z_0] = \{1,2,3,4\}$
$[Z_1] = [\beta \wedge [next]\, Z_0] = [\beta] \cap Pre_A(next, [Z_0]) = \{1,2,4\} \cap \{1,2,3,4\} = \{1,2,4\}$
$[Z_2] = [\beta \wedge [next]\, Z_1] = [\beta] \cap Pre_A(next, [Z_1]) = \{1,2,4\} \cap \{1,3\} = \{1\}$
$[Z_3] = [\beta \wedge [next]\, Z_2] = [\beta] \cap Pre_A(next, [Z_2]) = \{1,2,4\} \cap \emptyset = \emptyset$
$[Z_4] = [\beta \wedge [next]\, Z_3] = [\beta] \cap Pre_A(next, [Z_3]) = \{1,2,4\} \cap \emptyset = \emptyset$

$T \models \gamma$? No because initial state not in γ

Ex Model check CTL formula $EF(AG(a \supset AXEX\, \neg a))$ showing its translation in μ-calculus



$[\alpha] = [EX\, \neg a]$
$[\beta] = [AX\, \alpha]$
$[\gamma] = [a \supset \beta]$
$[\delta] = [AG(\gamma)]$
$[\eta] = [EF(\delta)]$

$\alpha = EX\, \neg a \longrightarrow \langle next \rangle \neg a$ $\quad \{2,3\}$
$[\alpha] = [\langle next \rangle \neg a] = pre_E(next, [\neg a]) = \{0,1,2\}$
$\beta = AE\alpha \longrightarrow [\overset{next}{-}]\alpha$
$[\beta] = [[-]\alpha] = pre_A(-, [\alpha]) = \{1,2,4\}$
$[\gamma] = [a \supset \beta] = [\neg a \vee \beta] = [\neg a] \cup [\beta] = \{2,3\} \cup \{1,2,4\} = \{1,2,3,4\}$
$\delta = \underbrace{AG\, \gamma}_{Z} = \gamma \wedge AX \underbrace{AG\, \gamma}_{Z} \longrightarrow z = \gamma \wedge AX\, z \longrightarrow \nu z.\, \gamma \wedge [-] z$ gfp

$[\delta] = [AG\gamma] = [\nu z\, \gamma \wedge [-] z] = \emptyset$
$[z_0] = \{0,1,2,3,4\}$
$[z_1] = [\gamma \wedge [-] z_0] = [\gamma \cap pre_A(-, [z_0])] = \{1,2,3,4\} \cap \{0,1,2,3,4\} = \{1,2,3,4\}$
$[z_2] = [\gamma \wedge [-] z_1] = [\gamma \cap pre_A(-, [z_1])] = \{1,2,3,4\} \cap \{0,1,3\} = \{1,3\}$
$[z_3] = [\gamma \wedge [-] z_2] = [\gamma \cap pre_A(-, [z_2])] = \{1,2,3,4\} \cap \{0\} = \emptyset$
$[z_4] = [\gamma \wedge [-] z_3] = [\gamma \cap pre_A(-, [z_3])] = \{1,2,3,4\} \cap \emptyset = \emptyset$ gfp

$\eta = EF\, \delta = \delta \vee EX EF\, \delta \longrightarrow z = \gamma \vee EX\, z \longrightarrow \mu z\, \delta \vee \langle - \rangle z$ lfp
$[\eta] = [EF\, \delta] = [\mu z\, \delta \vee \langle - \rangle z] = \emptyset$
$[z_0] = \emptyset$
$[z_1] = [\delta \vee \langle - \rangle z_0] = [\delta] \cup [pre_E(-, [z_0])] = \emptyset \cup \emptyset = \emptyset$ lfp

$\alpha = \langle - \rangle \neg a$
$\beta = [-]\alpha$
$\gamma = a \supset \beta$
$\delta = \nu z\, \gamma \wedge [-] z$
$\eta = \mu z\, \delta \vee \langle - \rangle z$

$T \models \phi$? = Is formula true in transition system?
More precisely in state 0 : $T_0 \models \phi$?
$0 \in [\phi] = [\eta]$? No $\longrightarrow$ formula not true in the transition system

AUTOMATA THEORETIC LTL MODEL CHECKING

Σ^ω = set of infinite words, infinite sequences of letters from alphabet Σ
Σ^* = set of finite words.
$L \subseteq \Sigma^*$ = finite language eg words ending with a
$L \subseteq \Sigma^\omega$ = infinite language eg words containing a finite number of a

Language recognition problem: determine whether a word w belong to language L → Use automata
We say $w \in L(D) \subseteq \Sigma^*$ if the corresponding run ρ in the automaton D ends in a final state

Nondeterministic Finite-state automaton N accepts $w \in L(N)$ if at least one run is accepting since more than one run is possible on some word.

$N = \{Q, \Sigma, I, \delta, F\}$ with:
Q = finite set of states
Σ = finite alphabet
$I \subseteq Q$ = set of initial states
$F \subseteq Q$ = set of final states
$\delta : Q \times \Sigma \to 2^Q$ = non deterministic transition function

UNION

Given two NFA $N_1 = \{Q_1, \Sigma, I_1, \delta_1, F_1\}$ and $N_2 = \{Q_2, \Sigma, I_2, \delta_2, F_2\}$, the union automaton $N_1 \cup N_2 = \{Q, \Sigma, I, \delta, F\}$ is defined as

$Q = Q_1 \cup Q_2$
$I = I_1 \cup I_2$
$F = F_1 \cup F_2$
$\delta(q, \sigma) = \begin{cases} \delta_1(q, \sigma) & q \in Q_1 \\ \delta_2(q, \sigma) & q \in Q_2 \end{cases}$

Intuitively, $N_1 \cup N_2$ chooses nondeterministically to execute either N_1 or N_2
Note: Q_1 and Q_2 must be disjoint

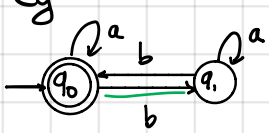
PRODUCT

In some way, $N_1 \times N_2$ is defined as

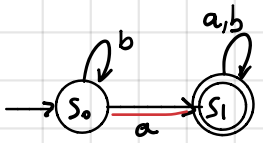
$Q = Q_1 \times Q_2$
$I = I_1 \times I_2$
$F = F_1 \times F_2$
$\delta((q_1, q_2), \sigma) = (\delta(q_1, \sigma), \delta_2(q_2, \sigma))$

$N_1 \times N_2$ executes N_1 and N_2 in parallel

Eg

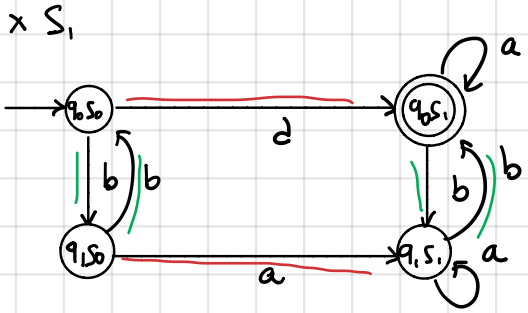


D_1 = recognizes even number of b



S_1 = at least one a

$D_1 \times S_1$



Th. For every regular expression α there exists a NFA N_α s.t. $L(\alpha) = L(N_\alpha)$
For every NFA N there exists a regular expression α_N s.t. $L(\alpha_N) = L(N)$

BÜCHI AUTOMATA

Deterministic (DBA) and Nondeterministic (NBA) Büchi automata are like DFA and NFA but they read infinite words $w \in \Sigma^\omega$
As there is no lost state in the corresponding runs ρ, the acceptance condition is to visit a final state in F infinitely many times

Th. The language $L = \{w \in \Sigma^\omega : w \text{ contains finitely many } a\text{s}\}$ can be recognized by a NBA but not by any DBA

Th. For a given NBA N, there exists a NBA $\overline{N}$ s.t. $L(\overline{N}) = \Sigma^\omega \setminus L(N)$

Union of two NBAs works exactly as for NFA.
Product $N_1 \otimes N_2$ called synchronous product is defined as:
$Q = Q_1 \times Q_2 \times \{1, 2\}$ ← bit to flag which automaton im focusing on
$I = I_1 \times I_2 \times \{1\}$
$F = F_1 \times Q_2 \times \{1\}$

$$\delta((q_1, q_2, 1), \sigma) = \begin{cases} (\delta_1(q_1, \sigma), \delta_2(q_2, \sigma), 1) & \text{if } q_1 \notin F_1 \\ (\delta_1(q_1, \sigma), \delta_2(q_2, \sigma), 2) & \text{if } q_1 \in F_1 \end{cases}$$

$$\delta((q_1, q_2, 2), \sigma) = \begin{cases} (\delta_1(q_1, \sigma), \delta_2(q_2, \sigma), 2) & \text{if } q_2 \notin F_2 \\ (\delta_1(q_1, \sigma), \delta_2(q_2, \sigma), 1) & \text{if } q_2 \in F_2 \end{cases}$$

$N_1 \otimes N_2$ switches the index every time a final state is hit

A language is called ω-regular if it is the union of expressions of the form $\alpha \cdot \beta^\omega$ with α and β being regular languages

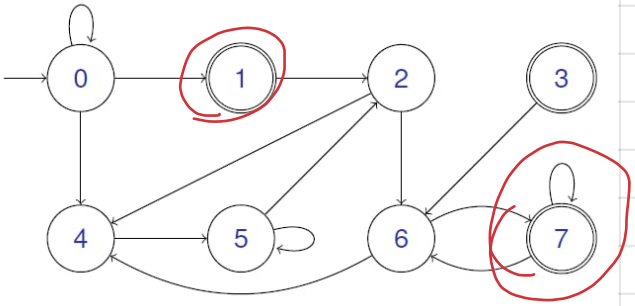
NONEMPTINESS

Given an NFA N decide whether $L(N) \stackrel{?}{\neq} \emptyset$ ⇒ Does a word w accepted by N exist?

↳ w is accepted by N iff exists a run whose path start from 0 and ends in a final state

⇒ Nonemptines ⇒ Reachability

REACHABILITY WITH FIX-POINT THEORY



$F = \{1, 7\} \subseteq Q$

$Z = F \vee \langle next \rangle Z$ (= reachability)

$\Downarrow$

$\mu Z.(F \vee \langle next \rangle Z) = Reach(F)$

Given a NBA instead we not only have to find a path from 0 to final state but must visit the final state infinitely many times. → Recurrent reachability.

Using fixpoint theory:

$Y = Reach(F \wedge \langle next \rangle Y)$

$\nu Y (Reach(F \wedge \langle next \rangle Y)) \rightarrow \nu Y.(\mu Z((F \wedge \langle next \rangle Y) \vee \langle next \rangle Z))$

[A Generalized NBA (GNBA) is like a normal NBA except that $F = (F_1, F_2 \dots F_n)$
A run ρ in N is accepting iff it visits every F_i infinitely often]

Th. For an LTL formula φ we can construct a GNBA N_φ s.t $\mathcal{L}(N_\varphi) = \mathcal{L}(\varphi)$

To do this we need:

Fischer-Ladner Closure: for a given LTL formula φ, the FS-closure of φ denoted $cl(\varphi)$ is the set of subformulas of φ and their negations. Defined as follow:

- $\varphi \in cl(\varphi)$
- if $\psi \in cl(\varphi)$ then $\neg\psi \in cl(\varphi)$
- if $\psi_1 \wedge \psi_2 \in cl(\varphi)$ then $\psi_1, \psi_2 \in cl(\varphi)$
- if $X\psi \in cl(\varphi)$ then $\psi \in cl(\varphi)$
- if $\psi_1 U \psi_2 \in cl(\varphi)$ then $\psi_1, \psi_2 \in cl(\varphi)$

eg $\varphi = p \wedge ((Xp) U q)$

$cl(\varphi) = \{p \wedge ((Xp) U q), \neg(p \wedge ((Xp) U q)), p, \neg p, (Xp) U q, \neg((Xp) U q), Xp, \neg Xp, q, \neg q\}$

A set $\alpha \subseteq cl(\varphi)$ is called atom if it is maximally consistent. that is:

- $\forall \psi \in cl(\varphi)$ either $\psi \in \alpha \vee \neg\psi \in \alpha$
- $\psi_1 \wedge \psi_2 \in \alpha$ iff $\psi_1, \psi_2 \in \alpha$

eg $\varphi = p \mathcal{U} q \quad cl(\varphi) = \{p\mathcal{U}q, \neg(p\mathcal{U}q), p, \neg p, q, \neg q\}$

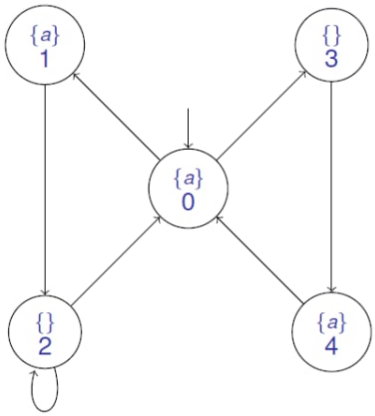
→ Atoms: $\alpha_1 = \{p\mathcal{U}q, p, q\}$ $\alpha_2 = \{p\mathcal{U}q, p, \neg q\}$ $\alpha_3 = \{p\mathcal{U}q, \neg p, q\}$ $\alpha_4 = \{p\mathcal{U}q, \neg p, \neg q\}$
$\alpha_5 = \{\neg(p\mathcal{U}q), p, q\}$ $\alpha_6 = \{\neg(p\mathcal{U}q), p, \neg q\}$ $\alpha_7 = \{\neg(p\mathcal{U}q), \neg p, q\}$ $\alpha_8 = \{\neg(p\mathcal{U}q), \neg p, \neg q\}$

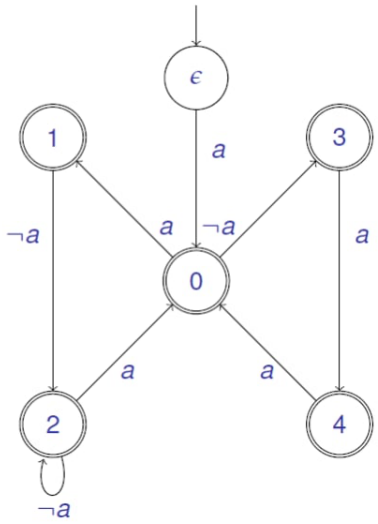
A state α (eg α_1) gives information on which subformulas of φ need to be satisfied when computation start from α itself (for α_1 $p\mathcal{U}q \wedge p \wedge q$ must be true)

Every atom containing φ is an initial state ($\alpha_1, \alpha_2, \alpha_3, \alpha_4$)
To move from atom α to α' ($\sigma \in \Sigma = 2^{Prop}$) i.e $\alpha' \in \delta(\alpha, \sigma)$ we have to check:
① $\sigma = \alpha \cap Prop$ → move only if you read something consistent
② $X\psi \in \alpha$ iff $\psi \in \alpha'$ → from state with $X\psi$ we can reach only states with ψ
③ $\psi_1 \mathcal{U} \psi_2 \in \alpha$ iff either $\psi_2 \in \alpha$ or both $\psi_1 \in \alpha$ and $\psi_1 \mathcal{U} \psi_2 \in \alpha'$
↳ in this case we can move wherever we want because we "verified" α

Final States: every subformulas $\psi_1 \mathcal{U} \psi_2$ in α holds on acceptance and thus contributes to F with the set $F_{\psi_1 \mathcal{U} \psi_2} = \{\alpha \in Q : \psi_2 \in \alpha \text{ or } \neg(\psi_1 \mathcal{U} \psi_2) \in \alpha\}$

FROM LABECED TRANSITION SYSTEM TO NBA





For a given LTS $\mathcal{T}$ and an LTL formula φ, Model Checking is the problem of verifying that all executions of $\mathcal{T}$ satisfy φ. ⟹ $\mathcal{T} \models \varphi \iff \mathcal{L}(\mathcal{T}) \subseteq \mathcal{L}(\varphi)$

- $\mathcal{T} \rightarrow N_{\mathcal{T}}$ $\quad \mathcal{L}(\mathcal{T}) = \mathcal{L}(N_{\mathcal{T}})$
- $\varphi \rightarrow N_{\varphi}$ $\quad \mathcal{L}(\varphi) = \mathcal{L}(N_{\varphi})$
- $\mathcal{L}(\mathcal{T}) \subseteq \mathcal{L}(\varphi) \iff \mathcal{L}(N_{\mathcal{T}}) \subseteq \mathcal{L}(N_{\varphi}) \iff \mathcal{L}(N_{\mathcal{T}}) \cap \mathcal{L}(\overline{N_{\varphi}}) = \emptyset$

! Can we avoid the need of complement of a NBA?

→ $\mathcal{L}(\overline{N_{\varphi}}) = \mathcal{L}(N_{\neg\varphi})$