## Rules with predicates

**Dealing with Objects**

Propositional logical can be see as dealing with one object at the time

But often we want to relate objects (maybe implicity)

Some of the objects are not explicitly stated.

*Ana is a professor*

we are saying something about a specific person, this knowledge doesn't say anything about bob

*Ana supervises Bob*

Two individuals in a relationship, something that we can’t do in propositional logic.

*A grandparent is the parent of a parent*

we are talking of a property of another individual, we know the property but you don’t know who is his child

*Uncle have siblings who are parents*

**Goal**

Extend rules to handle this “predicate” knowledge

uncle(x) ← sibling (x,y), parent(y,z)

more complex expression

x is an uncle, if x have a sibilin y and y is the parent of z

Interested in a property of an object x, it is a general rule, if I can prove that x have a sibilin and that sibilin is a parent I immediately know that that object is an uncle

we must dive into predicate logic

**(Restricted) Predicate Logic**

From propositions to predicates

In propositional logic we have the property mammal

In predicate logic we can’t say just mammal (have no sense), we need to give an argument

**mammal is a property**

mammal(dumbo)

mammal(phineas)

this expression have a true value

**dumbo and phineas are object** and by themself they don’t have a value

Just saying mammal have no sense, to have a truth value we need to have a whom

IsParent(ana, bob) in this case we have a predicate that puts together two objects, can say that ana is the parent of bob, and can be true or false

parent is a binary predicate, using it with one argument have no sense

**Introducing variables**

We want to have placeholders for constants

parent(ana, x)? logically makes no sense, depends on what x stands for, depends on what x stands

the truth value of this expression depends on the value of x

In predicate logical we need to give a scope to this variable x

∃x.parent(ana,x) there exist an x such that ana is the parent of x, we can find a predicate that begin true

∀ x.parent(ana,x) for all possible x ana is the parent of x, any constant makes it true

**Syntax of Predicate Logic**

We have arbitrarily many…

* constans C: a,b,c…
* unary and binary predicate symbols P: P, Q, R… (some of them have 1 arguments, some 2 arguments)
* variables V: x,y,z…

A predicate is P(t) or Q(t,s)   
P is a unary predicate symbol

Q is a binary predicate symbol

A literal il predicate or its negation

A predicate Horn clause is a disjunction of literals with exactly one positive literal

**Example**

P, R, unary

Q binary

P by itself is not a predicate, we need an argument

Q(x) is not a predicate need two arguments (is binary)

P(b) is a predicate

Q(x,a) is a predicate, we have a binary symbol with a variable x and a constant *a*

Clause (in rule form)

Q(x,y) ← Q(b,y), Q(y,z), P(x), R(z)

*[x and y are in the relation Q, if b is in the relation Q with y, and y is in the relation Q with z, and x belongs to P and z belongs to R]*

We don’t know how to interpret those rules

**Semantics**

Predicate represent properties of objects (**unary**) and relationships between objects (**binary**)

P(a) is true than the object *a* has the property P

Q(a, b) is true than a is in the relationship Q with b

The **two arguments in the predicate have an order**, we can’t put them backwards, and will have a different statement. Some statements are simmetrics like IsSiblin(a, b) is the same as IsSiblin(b,a) but those are specific kind of relationship

Proportional valuations are not enough

We must specify which objects satisfy which properties and relations

**Interpretation-based Semanticas**

An interpretation has two components:

* domain : what are we talking about constains but they can be created with arbitrary names. Says which world we are talking about. Can be like the students of a university, refers to the objects of the interest
* interpretation functions: tells us how to read the symbols in this domain we are talking about.
  + ai € Δi if the constant is the immatriculation numbers of the students of an university the interpretation of that immatriculation number is the person
  + if P is unary Pi ⊆ Δi if we have a unary predicate symbol (P) that refers to a property it is a set of objects
  + if P is binary Pi ⊆ Δi × Δi is a pairs of objects

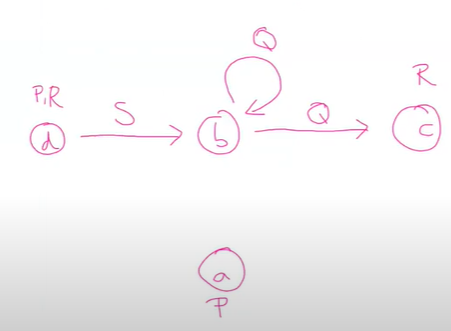
We interpret:

* constans as objects
* unary predicates as classes of objects with the property
* binary predicates as relations between objects

**Interpretation as Graphs**

A graph is a collections of points (nodes) and arrows (edges) connecting them

Interpretations can be equivalently expressed using labelled graphs



This represent an interpretation, the domain are the objects a,b,c,d

Δi = {a,b,c,d}

Pi = {a,d}

Ri = {c,d}

R(c) is true in this interpretation

Qi = {(b,b), (b,c)}

Si ={(d,b)}

**Predicate Rules**

A predicate rule is a predicate clause

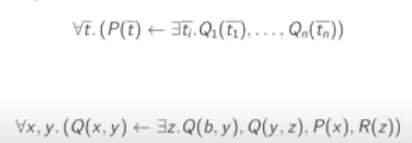


such that: all variables in P() appear in the body

A predicate knowledge base is a finite set of predicate rules

When are they all true?

**Formalisation**



for all possible value P(t)

∀ x,y.(Q(x,y) ← ∃z. Q(b,y), Q(y,z), P(x), R(z))

take two arbitrary objects (x,y) if you can find an object z such that all the predicates are true than Q(x,y) has to be true

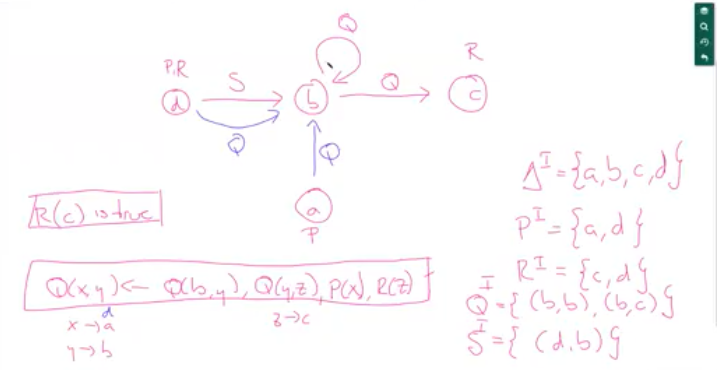
What does this mean?

Q(x,y) ← Q(b,y), Q(y,z),P(x), R(z)

take any value of x and y if you can make true this body you can make true Q(x,y)

x→ a

y→ b



we are interested not in all the possible interpretation but only that satisfied our rules

**Models**

The interpretation I satisfies the rule



iff whenever I makes the body true, it also makes the head true

I is a model of the KB K iff it satisfies all rules in K

K ⊨ P() idd P() is true in all models of K

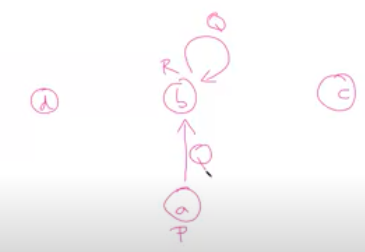
**Example**

Q(x,y) ← Q(b,y), Q(y,z), P(x), R(z)

Q(b,b) ←

R(b) ←

P(a) ←



Lesson 6 - 14/10/2021

**Knowledge Propagation**

We can propagate the knowledge building a **minimal representation** of all possible models (canonical model)

Present a knowledge propagation algorithm extended to the predicate case, this case is more complex, it have to do more work, but the idea of the algorithm is exactly the same

We are trying to find things that follow whenever our knowledge is true, and know what kind of facts are true in all possible models of our knowledge base.

Know what kind of facts are true in all possible models, try to build a minimal representation of all the possible models (canonical model) a small model that have the minimal constraint, enclude all the knowledge, **if we can conclude something for this model it is true, than is true for all possible models**

Minimal not in term of size but in term of the constraints (vincoli/ limiti), it is the weakest model part of this language it that ∃ a model like this

This canonical model can be **queried** (interrogato) to decide entailments (all the possible conclusion)

Canonical model depends on the object that we have, it is created based on the individuals that we have explicitly stated, if we add new individual this might be not a model anymore

**Canonical model Construction**

we are trying to build a label graph, we have only unary and binary elements, we have notes and labels on the notes

The canonical model is built from an empty model (graph) by adding only the necessary ingredients, to satisfy the constraint of our models but nothing more

We guarantee that is a model by satisfying all the constraint

Three steps:

1. **create all relevant models** (constants). Need to have some notes, look in the knowledge base of all the constants, we will assume that each constant refer to a different element of the domain
2. **add facts**. in the case of the propositional case. Once that we know which are the objects we add the facts
3. **propagate rules**. using the facts try to apply rules to this fact, if we can make truth the body of our rule that we add everything needed to satisfy the head, if the body is true the head must be true.

**Steps in more detail for the canonical model**

1. **Relevant nodes**

The **domain** of the canonical model is the set of all constants that appears in the Knowledge Base (KB).

They are interpreted as themselves (standar name construction). Formally an interpretation takes a domain and than says that each constant of the KB is mapped truth the intepreration function to an element of the domain, the domain have the same set so we map them to themselves

The object are named in the KB with the standard name (real name that they have in the world)

1. **Fact encoding**

Remember that we only have unary and binary predicates (predicates that apply to 1 or 2 elements)

For each fact P(a) ← add the label P to the node a

for each fact Q(a,b) ← add a Q labelled edge from a to b

1. **Rule propagation**

Take a rule P(t) ← Q1(t1),…, Qn(tn) if all those predicated are true that the predicate in the head must be true

With variables X:= {x1,…,xk}

Substitute the variables with constraints, once they are substitute they are facts

Look at all the possibles substitution

For each substitution of σ of x do

If σ (ti) € Qi for all i then encode the fact P(σ (t))

Repeat until we can’t add no possible knowledge

**Example**

We have 4 facts: with 5 objects

P(a) ←

P(e) ←

Q(a,b) ←

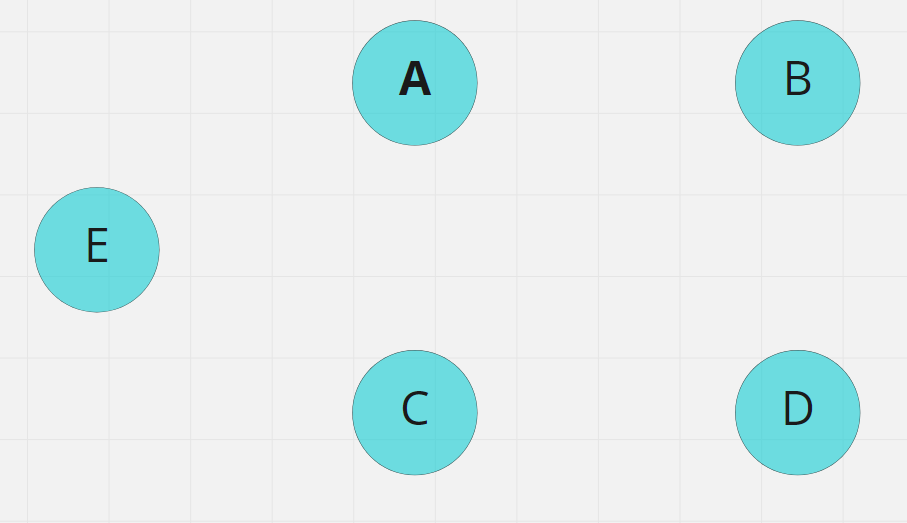
Q(c,d) ←

We have 2 rules:

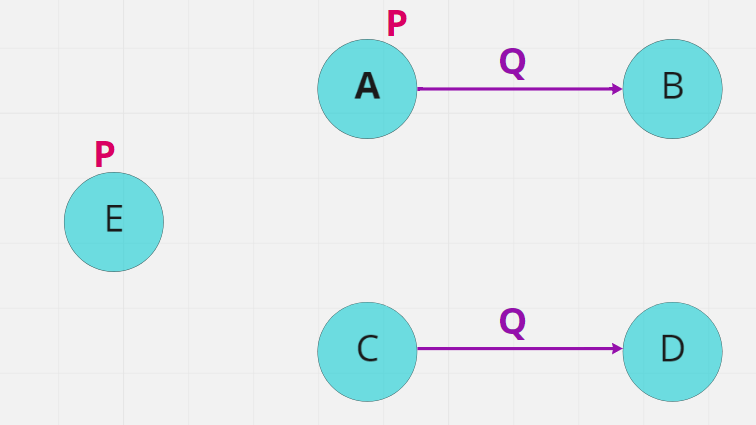
P(x) ← Q(y,x), Q(x,z)

Q(x,y) ← P(x), Q(y,z)

First step: build a note for each object that appears in the KB

****

Second step: encode all the facts that you have



Third step: propagate knowledge truth the rules

* First rule:

P(x) ← Q(y,x), Q(x,z) if I have Y -q→ X -q→ Z than the middle element (X) must be label with P. We don’t have a situation like this, we can’t apply this rule

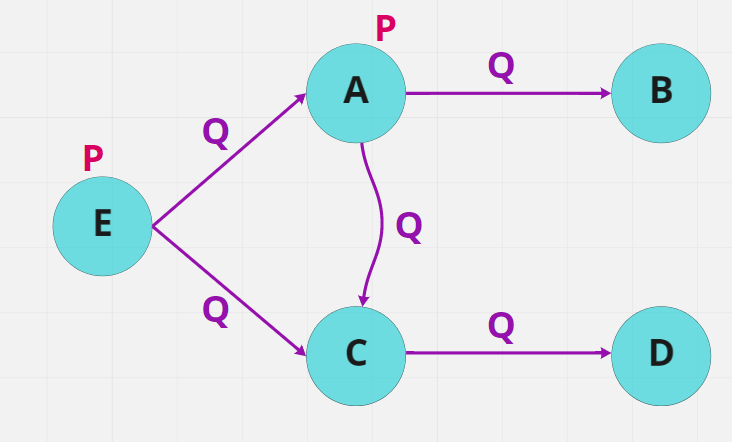
* Second rule:

Q(x,y) ← P(x), Q(y,z) if I have a element annotated with a P and an edge from y to z than I should have an edge from the P element to y

e is annotated with a P Q is (a,b). add Q between e and a, we have to add it otherwise it will not be a model, we need to satisfy all the rule

also add q edge between e and c

with x = a add q between a to c

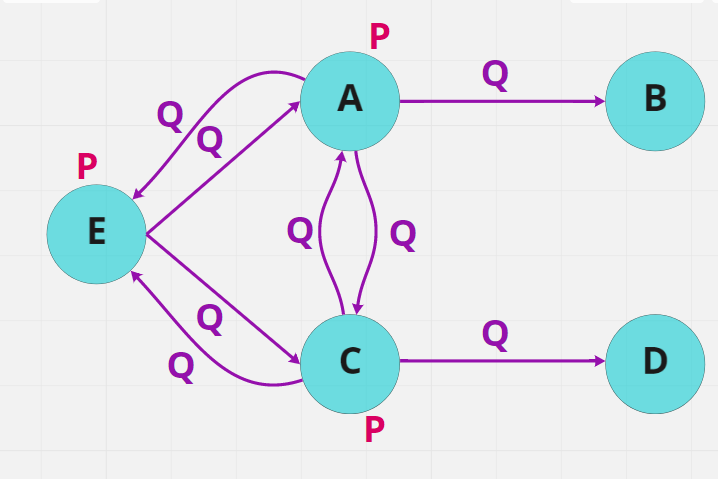


go back first rule, that originally I couldn’t use

P(x) ← Q(y,x), Q(x,z)

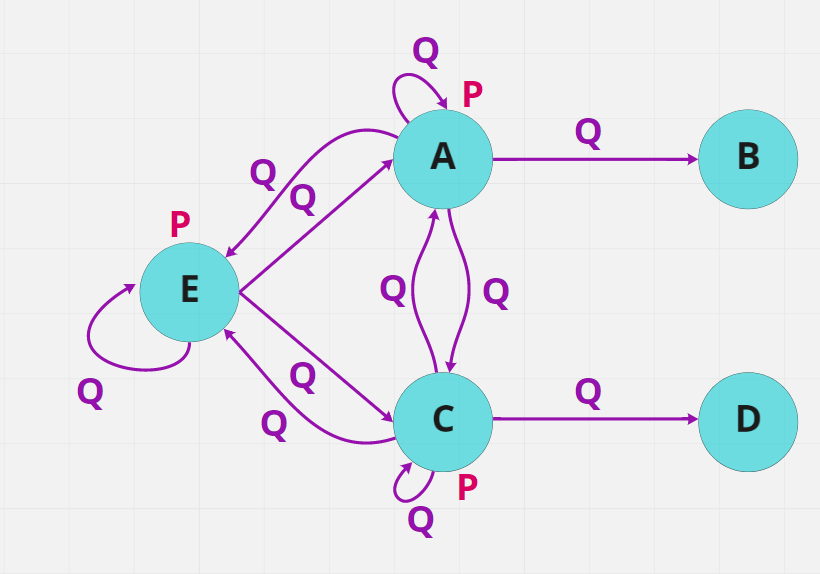
Now I can use it, and I can add a P to c

also x = c add q between c and a (second rule)



Apply all the rules

When we substitute we can give the same value



**Canonical Model**

The resolutions interpretation Ican is a model of KB

By construction, all clauses in the KB are satisfied

Called the canonical model (we will see why), it it the weakest model

An interpretation is a model if it satisfy all the rules

**Reasoning from the canonical model**

A fact P(a) is entailed by KB K iff the canonical model Ican of K satisfies P(a)

(If the corresponding label/edge appears in the graph)

If we want to know whether a fact is entailed in the KB we just have to check with this canonical model satisfy that fact

**Query answering**

Suppose that we want to know **all** the objects with a given property

Find all a such that K ⊨ P(a) (query p(x))

Simple graph exploration

P(x) ti will give back a, c, e because they are label with P

**Correctness**

To show correctness we must guarantee that the method is

* **complete** every consequence is found, show that our method is complete. Everything that follows is derived
* **Sound** every fact follows from all models. Everything that is derived follow from the KB

**Completeness**

If Ican does not satisfy P(a) then KV= (not a consequence)P(a)

If K entail V= P(a) than P(a) will appear in the canonical model

P(a) will appear in the canonical model

Why?

A is a consequences of KB is something that follows from all possible model for the KB

I take 1 specific model (canonical model) if that model does not satisfy this fact I am sure that not all model satisfy that fact

If I find one model that does not satisfy that property, that the property can not follow from all possible model of KB so it not a consequence of KB

The main point is that Ican is a model, and being a conseguenze ed of the KB is something that follows from ALL possible models of KB

Ican is a model

**Soundness**

Soundness is messier, proving it is more complex.

We must show that if Ican satisfies P(a) then all models satisfy P(a)

Whatever follows from the canonical model must follow from all possible model, if the canonical model says yes all models must say yes

The point is to prove the canonical model in an intelligent way

Try to make the model the least restricted model

We show that every model I “contains” (is not really an inclusion) Ican

Whatever Ican say is also stated by I but I may say more (can add additional constraint…)

**Homomorphism**

Take a model and Δ (set of all the constant in the KB) be the set of all constants in K

is the domain

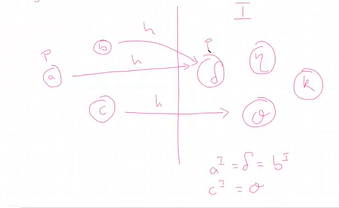
We define h: by h(a)= ai a € Δ

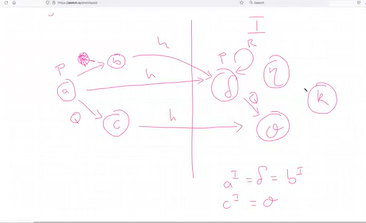
Maps every constant to an element of the domain

We show that **for all** a, b € Δ:

If a € PIcan then h(a) € Pi (Ai  € Pi)

If (a,b) € QIcan then (h(a), h(b)) € Qi  ((ai € pi) € Qi )





**Proof by induction**

We follow the steps used to construct Ican

Canonical model was constructed with an interaction, starting with facts and adding more facts

Proof by induction is similar, start with the things that are true, make sure that the proprieties are also true

For the facts it holds trivially by construction (because I is a model)

If P(a) was added through a substitution δ on

P(t)←Q1(t1),.., Qn(tn)

By induction h(δ (ti)) € Qii

But I is a model thus h(δ(t)) € Pi

We construct the graph following some steps, whenever we apply those steps we keep the property that we want to prove

If it true from the body of the rule than must be true for the head

**Conclusion**

It suffices to look at the canonical model to derive all atomic consequences of a Knowledge Base

**Complexity**

Each fact can be added at most once

How many steps we need to do

1. Take our graph
2. We know how many objects there are
3. Add unitary or binary facts (labels to notes to between notes)

To add a fact we must check **for each rule** and **for each substitution** whether e the body holds

If the KB has k constants and a rule has n variables the there are **k^n** subistitions

If there are m rules with at most n variables we need to check at most m\* k^n checks

The more variables the more checks

————

**More complex queries**

We constructed the canonical model, process that allows us to derive all the simple consequences of our KB either unary or binary

It is limited, often we would like more complex properties

What if we are interested in complex properties?

For example find all pairs (x,y) such that P(x) and there is a Q-path from x to y

We need to encode in some way the knowledge of Q-path, we first need to defined it

QPath(x,y) ← Q(x,y)

QPath (x,y) ← Q(x,z) QPath(z,y)

Query P(x), QPath(x,y) but is it a complex query, need to satisfy two properties

To do so, add another rule: R(x,y) ← P(x), Qpath(x,y) (R(x,y))

pushing the complex property inside a rule, as long as we know how to satisfy a complex property just using rules

**Limitation and generalisation**

As presented we can only query properties or amity 1 or 2

Technically there is nothing special in these arities we can easily generalise to n-ary predicate but we lose the graphical representation

The terminology get more complex, in technical terms we can

all the construction of the language can be redone using predicated with unary or binary, the big issue is that we lose the graphical representation

**How expressive are rules**

Predicate rules were introduced to handle individuals and their relations, want to speak about object and the relations between those objects

Constraint: all variables in the head of a rule appear also in the body, we have variables in rules (can be in the body or in the head), all the variables that appear in the end need to appear in the boy too

**3. Rule propagation**

Take a rule

P(t) ← Q1(t1),…,Qn(tn)

With variables x:= {x1,…, xk] (in the rule)

For each substitution δ of x do

If δ (t1) € Qi for all i then encode the fact P(δ(t))

**What are we doing?**

When build canonical model

To decide whether we can add a fact to Ican we:

1. **Instantiate** the variables in the rule. make the substitution
2. Check the application of the instantiated variables

What if we instantiate all rules first?

**Grounding**

The **grounding** of a rule is the set of all its instantiations given a fixed set of constans

Take the rule with variables and the ground is a set of rules with no variables

The grounding of a KB is the grounding of al its rules

**Example**

P(a) ←

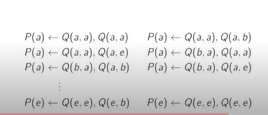
P(e) ←

Q(a,b) ←

P(x) ← Q(y,x), Q(x,z)

The ground is the set of all the rules that arise from all the possible substitutions of the variables, 3 values for x, 3 for y and 3 for z

27 evaluations



The grounding is the expansion of all the rules with all the possibles constants, making explicit what we are doing when building a canonical model

each expression is either true or false

**Propositional KB**

The grounding of a prudiate KB is a propositional KB with elaborate propositional variables

The predicate representation is much more compact