In propositional logic can’t do syllogism

*All men are moral*

*Socrates is a man*

*————*

*Socrates is mortal*

Can’t write this, this is an internal stricter can’t be realised in propositional logic. In first order logic it is easier to formalise it, write much less code

**First order logic**

Predicates: *Paul loves Clara*. Loves need 2 arguments loves(x,y) - binary predicates

*Paul gives Claudia a rose*

Gives (x,y,z) need 3 arguments - thermal predicates

Functions, constants (called also proper names)

Quantifies (*all, some*…)

Predicates

You **can say more**, in first order logic, means that you have **worse computational performance**. In order to get reasonable performance, or restricted first order logic (make it more powerful than propositional logic) but no so powerful to become intractable

Another application is to try to consider fragments associated with decision procedures.

Take existing decision procedures then come to the decision domain (artimic, computer science…) and bring them together to express fragments of first order logics.

SMT: satisfiability modulo theories

Satisfiability: want to use what we know from SAT

Theories: knowledge coming from specific domains (arithmetic, computer science, logic…)

Applications: planning (I know what to do for a certain goal), verification (check that a code is correct, no bugs), constraint solving (pigeon problem…) … AI applications

To do so start with first order logic

Reduce more complicated problem to SAT

**First order logic**

Chapter 4

First order language also called elementary language

First order = elementary

first orders: Quantifiers are limited to a range from individuals

I can say there exist something

But I **can’t** say there exists a function, or for all property, for all functions. This is high order logic.

First order logic focuses on objects. If you increase the expressing power you decrease the performance. This depends on the application

Elementary logic need three things

L = (P, F α )

P = predicate symbols. set of predicates, *to be less then to love, to give*, something than need some arguments

F = function symbol.

α= is a function from P ∪ F → IN functions giving the number of arguments

**Predicate symbols**

Common noun: *dog, man…* this is a predicate with one argument. Man is the set of man, dog is the set of dog

Intransitive verbs: *sleep*, it is a unary predicate. Sleep is the set of people that are sleeping

Transitive verbs: *love…* is a binary predicate, needs two arguments. Binary predicate

Dative verbs: *give* is a thermal predicate. Predicate symbols.

**Functions symbols**

log(x) successor(x) are unary function symbols

+ and \* are binary functions symbols, require two arguments

**α** **arity function**

α (log) = 1

α (+) = 2

α (give) = 3

α (sleep) = 1

α gives the number of an argument

**TWO SPECIAL CASES**

α (f) = 0 arity zero

We can see mathematical constants, like *0, 1, π , e…* as functions with no arguments. The same for proper names, Paul, Claudia…

α(P) = 0 P is a predicate symbol. For impersonal verbs α(*rains*)=0

Predicate symbols of arity zero are just our well known propositional variables. First order logic is an extension of propositional logic. In propositional logic no function symbols and no predicates have no arguments

In propositional logic we have one kind of entities

Here we have

* terms. denote individuals.
* formulas. denote booleans (truth values)

Tern: *(x+5)\*y* it denote a number

Formula: *x>0* it has no definite meaning, but when you know **x is either true or false**

In first order logic you can use variables as individuals.

In first order logic we are using individual variables. Which are names for unspecified objects.

29.10.2021

**Notes 4.2**

**First order language (signature)**

L = (P, F, α )

F and P are disjoint sets!

* term: indicate an object
* Formula: indicate a truth value

Definition of the set of terms ( L fixed)

* every variable xi is a term. Variable are a countable set, but I will use a finite set of them, in principe there are infinitely many variables
* Every constant (c € F α ( c )= 0) is a term
* If f € F and α (f) = n and t1,…tn so is f(t1,…tn)

Infix notation: Log(x)+1 ← practical use

Prefix notation: +(log(x),1) ← definition

There is no real difference, in SMT-LIB notation (z3) → (+ (log x) 1 )

**Definition of the set of formulas**

If P € P (predicate symbol), α (p) = n and t1..tn are terms, then

P(t1…,tn) is a formula

If A,B are formulas so are (A^B) (AvB) (A → B) ( ¬ A)

If x is a variable and A is a formula ( ∀ x A), ( ∃xA) are formulas

**List of convention and definition**

elimination of brackets ) (

¬ ∀ ∃ precedence over ^ v ^ v precedence over →

¬ ∀ xP(x,y) → Q(x) v R(y) is ((¬ (∀ xP(x,y))) → (Q(x) v R(y)))

A term is **ground** when there are no variables in it

3+2 is ground

3+x is not ground

**Occurrence**: is place where the variables/formulas is

An occurrence of a variable x in a formula A is **bounded** if it is **located inside a subformula** of the kind **∀ xB or ∃ x** (that start with a quantifier). Otherwise is free

Every formula can be build in an unique way

∀ x(R(x,y) → Q(x)

(∀ x(R(x,y)) → Q(x) first add brackets

In this formula x have two occurrences (∀ x is not counted) an occurrence is a place where the variable is

R(x,y) is bounded. Located in a formula that start with a quantifier (∀ x)

Q(x) is free

y have one occurrence and it is free, because they quantified is x not y

A(x1,x2,…,xn) t(x1,x2,…,xn)

at most x1,x2,…,xn have a free occurrence in A

What letters stands for:

A,B,C… formulas

t,u.. terms

x,y,z… variables

c,d… constands

f,g,h…. Functions

P,Q,R… predicates

**POSSIBLE QUESTION IN EXAM**

Suppose I have a language L = (P = {P,R}, F ={f,c}, α )

α (P) = 1 α (R) = 2 α (f) = 1 α (c) = 0

Are these terms:

f(x) yes P(x) no (is a formula) f(x,y) no (not correct), f have arity 1 not 2

Show the free and bound occurrences of variables in

(∀y(R(y,x)) → ¬ P(f(y))

α (P) = 1 α (R) = 2 α (f) = 1 α (c) = 0

Bound y in R(y,x)

Free x in R(y,x)

Free y in f(y)

∀x(R(x,f(x)) → R(x,y))

Bound and Free

Give me two examples of a ground term in L

Ground term:

Constant c is correct

f(x) or f(f(c)) or f(f(f(c)))… are correct

x, y f(x) are **not** ground

**Sentence**

A formula where **no** variable occurs free

∀ xR(x,y) ∀ x(R(x,f(x))

Not sentence sentence

Because free variable y

∀ xR(x,x) → P(x) if I put the brackets (∀ xR(x,x)) → P(x) is not a sentence because P(x) is free

**Universal closure:**

A(x1,x2,…,xn)

A∀ ∀x1..∀xnA

(∀xR(x,y)) not a sentence, y is a free occurence add ∀y

∀y(∀xR(x,y)) Universal closure of (∀xR(x,y))

A∃ ∃x1.. ∃ xnA ∃ ∃y∀xR(x,y) ∃ closure of ∀xR(x,y)

**Substitution**

t(x1…xn) t1(u1/x1, ...un/xn) u1 replacing x1

Term obtained by replacing x1 with u1… xn with un

The replacement is **simultaneous** (otherwise can get different situation)

Suppose g function symbol of arity 2

g(x,y) want to replace x by f(y) and y by constant c

g(f(y)/x, c/y) g(f(y),c) is the result

If the replace is **not** simultaneously I get g(x,y) g(f(y),y) g(f(c),c) is not the same. The replacement one after the other get a wrong situation, must be simultaneous

A(x1,…,xn) want to replace x1 by t1 and xn by tn simultaneously

A(t1/x1, …, tn/xn) **only free occurrence are replaced** the bound one are not replaced

We want to avoid “clashes” (conflicts)

∃y R(x,y)

Want to replace x by y the result **can’t be** ∃y R(y,y) not acceptable

The point is that in the replacing term (“y” there is a variable that occurs bounded).

∃ y R(x,y) first rename the bound variable. Just change the name *y = z*. Change the name have no real meaning

∃ z R(x,z) then I make substitution

∃ z R(y,z)

at any step you can always rename bound variables so that the variables occurring bounded (in your f formula, your problem...) are disjoint from those occurring free

It is good practice! very recommended

10 nov. 2021

L = (F, P, α ) set of function symbol, set of predicate symbol and arit functions

L-structure for the language L (related to the language)

Structural = domain, and interpretation function

*A* = (A, *I*)

*A* = calligraphy letters, structure

A = domain, corresponding bold letter

*I* = interpretation function

**Domain**: set of objects we are talking about (set of numbers, set of students… usually the domain is assumed to be not empty)

**Interpretation function**: what is the meaning of the symbols in that domain

“Black haired” it become a set of objects formed by the set of people that have black hair

If I interpret it become a real

“black haired” ⊆ A

The interpretation of a symbol is the meaning that that symbols have in that domain

*I*(f) is the meaning of f in the domain A

The same symbol can have different meaning in different situation

*Nice* have different meaning if I am talking about girls/boys or cat

The meaning is given by the interpretation function

*I* is given a meaning to all symbols of the language L in our structure *A*

A2 is the set of ordered pair from A

A3 is the set of ordered triples from A

An is the set of Tuples of length n from A, that is the set of (a1,…an) where a1 € A, …, an € A

N-th Cartesian power of A

Cartesian product of a set with itself

A0 = {\*} set with only one element (it is a convention)

The interpretation

R € P α(R)= n

I(R)⊆ An

F € f α (f) = n

I(f): *A*n → *A*

The interpretation of a relation symbol is a relation, the interpretation of a function symbol is a function

If the function is declared with n arguments the domain of the function is *A*n

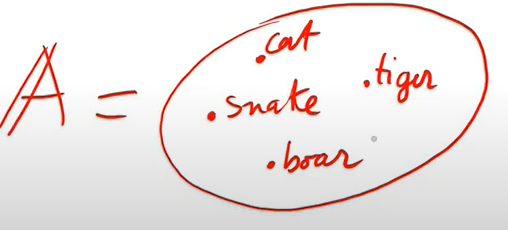
*“Nice”* is declared as a predicate with arity 1

Nice is an element of the set of pair P

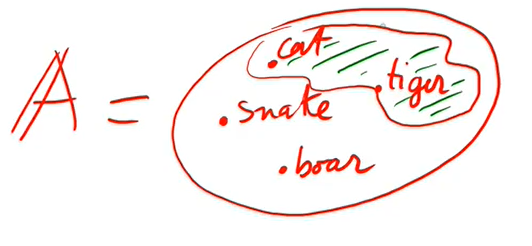
Nice € P α(nice) =1

My domain is composed by cat, snake, tiger, bear

A = cat, snake, tiger, bear



For Ghirlandi nice is cat and tiger I(nice) = cat, tiger



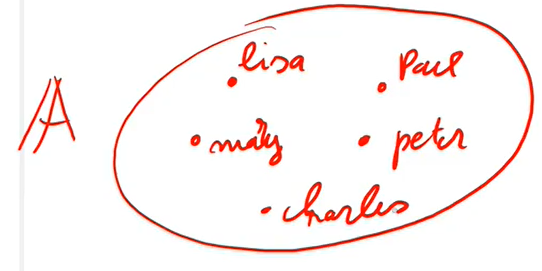
The interpretation of nice (I (nice) ) according to Ghilardi

“Loves” declared as a predicate symbol of arity 2

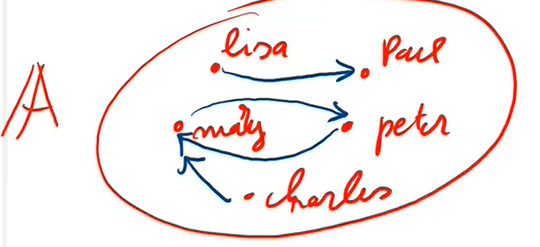
(Any transit verb have arity 2)

Love € P α(loves) = 2

Domain A: Lisa, Paul, Mary, Paul, Charles



I(loves) ⊆ A2



{(lisa, Paul),(Mary, peter),(peter, Mary),(Charles, Mary)}

the interpretation of a set of pairs is useful to draw with arrows

Since the verb is transitive is a set of pair, it is important to have an order

Language: formed by a set of functions, a set of predicates and a arity function

Structure: formed by a domain (that is not empty) and an interpretation (associates with predicates of arity n tuples, with function of arity n a function of n variables)

Understand that usually in common life we have a precise correspoing, in general this language is used to talk to a machine.

Ground terms: terms without variables

Sentences: formals without free variables

suppose I have a structure

A= (*A,* I)

With every ground term T I want to associate its “meaning” ( or its “evaluation”) in A

Ta € A

A =set of human being

F = “to be the father of”

A = Peter, Paul, Andrew, Charles, ...

I(f): Peter → Paul, Paul → Andrew…

Suppose that a ground term is ”(the farmer of (the father of (of peter)))”

f(f(p))a = Andrew is a ground term (no variables) so I can commute it’s value

Constant is a function symbol of arity 0

C € f α(c)= 0

I(c) € A

I(c): A0 → A

A0 is one element

Define the meaning of a ground term in a structure

f(f(p)) = I(f)(I(f)(I(p)))= I(f)(I(f)(peter)))=I(f)(paul)) = Andres

for a ground term t, we define ta € A

* if t is a constant ta is = I(t)
* if t is f(t1...tn) the evaluation is f(t1...tn)a = I(f)(t1a...tna)

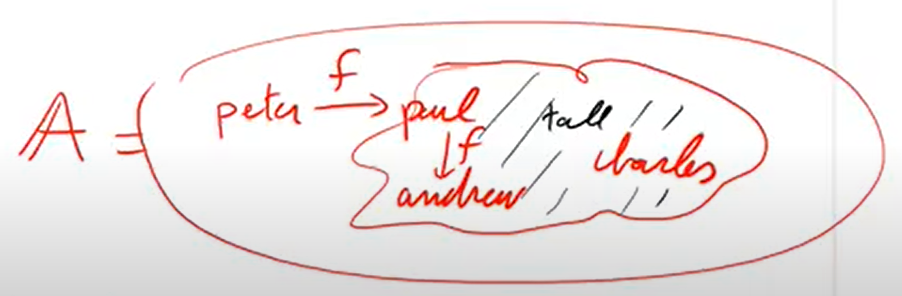
Ta the value of t in A

**Define the notation of true**

For sentence B we want to define when B is true in A

A = peter →paul→ Andrew

tall: set with elements paul, andrew and charles



The father of paul is tall

tall = T

father = f

paul = P

T(f(p)) when is this true?

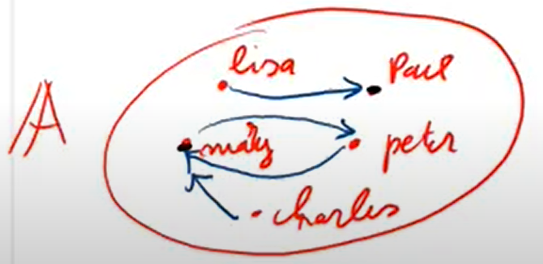
First I have the evolution f(p) and then ask whether f(p)a € I(p)

ok is true

Sentence peter is tall T(pe) evaluate Pei € i(t)

“Paul loves Mary” check if is true

(Paula, marya) € I(“loves”) is FALSE



Paul love lisa FALSE

Lisa love paul TRUE

**4.3**

L = (P, F α ) A(*A*, I)

For t ground, we define ta € *A*

We assume that L is expanded to a language La where we have a name for all elements in A

La is an ideal language

The name of a is written we assume of course that I()) = a

Better to make the expansion of the language

We define by induction for a sentence (formula with no free variables) B what it means that B is true in A

A ⊨ B B is true in A

B atomic P(t1,…tn)

A ⊨ P(t1,…tn) iff t1a…tna € I(p)

A ⊨ B1 ^ B2 iff (a ⊨ B1 and a ⊨ B2)

A ⊨ B1 vB2 iff (a ⊨ B1 or a ⊨ B2)

A⊨ ¬ B1 iff (a ⊭ B1)

A ⊨ ∃ xB1 iff (for some a € A A ⊨ B1(/x)

A ⊨ ∀ xB1 iff for all a € *A* a ⊨ B1(/x)

Nice = predicate of arity 1

near to = predicate of arit 2

Many = constant

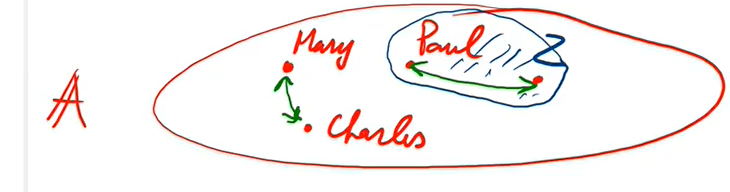
Paul = constant

Charles = constant

“Somebody near Mary is nice”

∃x(Near(Mary,x) ^ nice(x))

A: Mary, paul, Charles



A ⊨ ∃ x(N(x,M)^n(x))

Iff a ⊨ N(M,M) ^ nice(M)

Or a ⊨ N(M,C) ^ nice(C)

Or a ⊨ N(M,P) ^ nice(P)

Or a ⊨ N(M,Z) ^ nice(Z) add the fake name

Evaluate a ⊨ N(M,P) ^ nice(P)

is false because A ⊨ N(M,P) IS OK and A ⊨ n(P) is false (Mi, Pi) € I(N) and Pi € I(n) ok. but the orange is false So the statement is false

Evaluate: all people near Paul are nice

∀ x(N(P,x) → n(x))

∀ x( ¬N(P,x) v n(x))

A ⊨ ∀ x ( ¬ N(p,x) v n(x))

4 cases

iff A ⊨ ¬N(P,M) v n(M)

and A ⊨ ¬N(P,P) v n(P)

and A ⊨ ¬N(P,Z) vn (Z)

and A ⊨ ¬N(P,C) v n(C)

they should be all true because it is ∀

Try to evaluate one:

A ⊨ ¬N(P,Z) or A ⊨ n(Z)

Zi € I(n) ok

Is ok because one part of the or is ok

Try to evaluate one:

A ⊨ ¬N(P,M) or A ⊨ n(M)

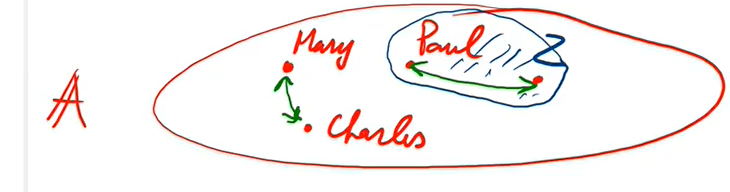
Iff A ⊭ not N(P,M) or A ⊨ n(M) it is false

(Pi, Mi ∉ I(N) since there is no arrow it is true

So it is true

**EXERCISE TO DO**

*There is some nice people near some nice people*



A ⊨ ∃ x ∃ y(n(x) ^ N(x,y) ^ n(y))

*All people have a nice neighbour*

∀ x ∃ y (N(x,y) ^ n(y)) is false

Esercizi svolti goodnotes 4.3

**4.3**

A ⊨ B(x1,…xn)? With free variables. Is like saying *he* is nice. Can be true or false, it is undetermined

There are two possibilities

Can be:

A ⊨ ∀ x1… ∀ xn or A ⊨ ∃ x1… ∃ xn

B is true B is satisfiable

Z3 and classical definition This is not standard

T set of sentences

A model of T is a structure A s.t A ⊨ B fo all B € T

A model makes everything true

The sentence B is a **logical consequence** of t iff B is true in all models of T

B is a **logical truth** iff B is true in all structures

A is a model of T is written

A ⊨ T

(structure on the left, set of sentences on the right)

B is a logical consequence of T is written

T ⊨ B

(set of sentences on the left, a sentence on the right)

B is a logical truth is written

⊨ B

(only a sentence)

B is true in A is written

A ⊨ B

(a structure on the left, a sentence on the right)

**Four uses of the same symbol**

Look at what is on the left and right, there is no ambiguity

**Alfred tarski**

12 nov 2021

L = (P,F, α) = € P

A = (A, I) I(=)= {(a,a) | a € A)

A ⊨ t1 = t2 iff (t1a ,t2a) € I(=) iff t1a = t2a

t1 t2 ground terms

A(x) x free variable in A

For every structure A A ⊨ t1 = t2 ^ A(t1/x) → A(t2/x)

A ⊨ t1 = t2

**SUBSTITUTION PRINCIPLE**

If two things are equal they have the same property

If *Charles = Peter* than both are yellow, or old, or tall… because they are the same thing

**Formulations problem in first order logic**

If you consider an arbitrary formula B checking whether B is a logical truth or whether B is satisfiable IS IMPOSSIBLE

No algorithm can do that, no program can solve this problem

Restricted classes of formula can get good results

**Restriction**

We can restrict the shape of B or restriction the kinds of structures we consider

If we make restrictions the problem may be solvable

**Case where B is ground**

Ground: no free variables in the formula

*Easy reduction to SAT (section 4.4)*

We associate with every atomic formula A a propositional letter PA

Different formulas are associated with different propositional letters

Replacing in B each atomic formula A with PA makes B a propositional formula (propositional abstraction)

R(a,b) → P(c) PRab → PPc

Consider every atom as a propositional formula

**Theorem**

A set of propositional formulas T is satisfiable in tasking semantics of first order logic iff its propositional abstraction is satisfiable (in the sense of propositional logic) - thus satisfiability of T can be checked by resolution, DPLL, CDCL,…

Formulas needs to be all ground

**Example 21**

Search *smullyan* for exercises

**Theorem**

T set of ground formulas

The propositional abstraction of T is satisfiable (meaning that has a model) iff T is satisfiable

T = {((a) vF(a), L(a)vL(b)vL(c),…¬ L(a)} … that this is unsat

P Q R S T ¬ R if this is unsat…

Try to reduce to a problem in propositional logic problem

Proof the theorem

If T is satisfiable so it is its propositional abstraction

There is a structure

A ⊨ all formulas in T V? V(all propositional formulas in T) =1

A ⊨ B → V(PB) = 1

The other way around:

Suppose that the evaluation

V( all propositional abreaction of formulas in T)= 1

A = (A,I)?

A = Herbrand universe = set of all ground terms

I(f) = (t1,..tn) = f(t1,..tn) ta = t the interpretation of a term is itself

F € fm

I(R) = { (t1,..tn) | V(PR(t1,..tn))= 1 } R € P α(p) = n

In this way

A ⊨ R(t1,..tn) iff (t1a,..tna) € I(R)

(t1,..tn) € I(R)

Iff V(PR(t1,..tn)) = 1

By induction it is immediate that A ⊨ B iff V (propositional abstraction of B ) =1

First order formulas, without identity, ground are “the same” as propositional formulas

You consider every atomic formulas as propositional variable

**EXERCISES**

L = (P, F, α)

P = { =, P, R}

α(=) = 2 = α(R)

α(P)=1

Y = {f,c,d}

α(*f*) = 2

α(c) = 0 = α(d) constants

**question**

*Term? Formula? Ground or not?*

[**Tabella**](https://docs.google.com/document/d/1AujBcQjdlI18nD7y9zCybshzRAYXJd7m1XXQuH8zkGc/edit?usp=sharing) **con cose da sapere**

f(x,d) term, not ground

f(d,d) term, ground

P(f(x,c)) formula, not ground

P(c,d) formula, ground

R(c) NO. R should have arity 2

R(c,d) → f(x) NO. A formula should imply a formula, not a term, also f arity 2

R(c,d) → P(c) Formula, ground

f(c,d) = f(c,f(c,c)) equality, need 2 terms, ground

f(c, f(c, f(c,d))) 2 arguments, both are term

P(x) → ∃yR(f(y,y),x) formula, not ground, not sentence

f(P(c),d) NO. P(c) is a formula, needs to be a term

**Atomic**: predicate letter applied to the right number of terms

**Literal**: negated atom or atom

**Clause**: disjunction of literals

**CNF**: conjunction of clauses

**Open formula**: formula without quantifiers

**Universal formula**: it is obtained from an open formula by prefixing some universal quantifiers

**Existential formula**: it is obtained from an open formula by prefixing some existential quantifiers

**EXERCISES 2**

L = (P, F, α)

P = { =, P, R}

α(=) = 2 = α(R)

α(P)=1

Y = {f,c,d}

α(*f*) = 2

α(c) = 0 = α(d)

[**Tabella**](https://docs.google.com/document/d/1AujBcQjdlI18nD7y9zCybshzRAYXJd7m1XXQuH8zkGc/edit?usp=sharing) **con cose da sapere**

¬ P(c) → R(x,d) Correct: yes - Formula: yes - Atom: no - Literal: no - Clause: y - Open: yes - Ground: no

∀ x(((R(x,c) → P(d)) → P(x)) Correct: yes - Formula: yes - Sentence: yes - Atom: no - Open: no - Ground: no - Universal: yes

**HERBRAND UNIVERSE**

The set of ground terms of the language

*c, d, f(c,d), f(d,c), f(c,f(c,d))...*

It can be infinite

It is finite when there are no function symbol (**only constants**)

If everything is ground you can limit yourself to the model built on the herbrand universe. If the herbrand universe is finite you have only finite remaining possibilities to check