We focus on a restricted class of formulas

**A clause is a disjunction of literals** (variables or negated variables)

X v Y v ¬ Z

¬ X v ¬ Y

X v Y v Z ¬ Y

Positive literals: X, Y

Negate literals: ¬ X, ¬ Y

A Horn clause is a clause with exactly one positive literal

Horn Clauses:

X

X v ¬ Y v ¬ Z

¬ X v Y v ¬ Z

NOT Horn clauses:

¬ X

X v ¬ Y v Z

¬ X v ¬ Y ¬ Z

**Equivalent Representation**

¬ X1 v ¬ X2 v … V ¬ Xn v Y

↑

What does it say?

(X1 ^ X2 ^ … Xn) → y

They are equivalent

If all X1…X2 are true then Y must be true as well

Y ← X1, …Xn

Y ← (Empty space, Y have to be true)

**Knowledge Representation**

A knowledge base is a (finite) set of rules

Platypus ← Monotreme, venomous, DuckBilled

Monotreme ← Mammal, Oviparous

Cetacean ← Mammal, Aquatic

Carnivorous ← Cetacean

LandDweller ← Monotreme

A knowledge base is a set of rules Clauses Formulas

As knowledge we consider them all true

My assumption is that all I wrote is true, all those rules are interpreted as formulas.

What other true statements follow from them?

A knowledge base is a set of formulas a1,…an

We consider them true, in other words we believe a1^…an is true

B is a consequence of this knowledge iff (a1 ^ … ^ am) → B is a tautology

X ← Y, Z, W

Y ← U, V

Z ← U

W ← Z, V

? X ← U, V

To do so, check U and V if they are true

U ← It is a fact that is true, I can remove it from others conditions

V ←

Remove the conditions that are true you will have X ←

So you will show that X is true

Lesson 4 - 4/10/2021 part I

Tautology: whenever all my knowledge is true then (a1 ^ … ^ am) → B must be true

The conjunction of all those formula must be true

very hard and long to check if it is a tautology.

**Notation**

K ⊨ φ K entails φ , φ is a consequence of K

K is a knowledge base

φ is a consequence that I am interested in

Whenever my knowledge (K) is true φ must be true

K → φ is a tautology

**Unit resolution**

In this language, we can do better, using unit resolution

In the case of rules, it is truth propagation.

**Intuition**

y ← X1, …, Xn

means that if ALL X1, … , Xn are true, than y is true

To guarantee that y is true, it suffices to show that each Xi (1≤ i ≤ n) is true. If you prove that from Xi to Xn all of them are true then for sure Y must be true.

In particular

Y ←

means that Y is necessary true

**Truth propagation**

Y ← X1, … , Xi-1, Xi, Xi+1, … Xn (Y is true if all the variables from X1 to Xn are true)

Xi ← (Xi is necessary true)

Since Xi is necessarily true I can remove it because I don't need to check it. Since Xi is true, we can “ignore” it in the body of the first rule

This simplifies the knowledge base, I have a smaller rule because Xi desappers

Y ← X1, … , Xi-1, *(̶X̶i̶)̶* , Xi+1, … Xn

If the facts arise, we can repeat the process, I might find a new variable that becomes necessarily true. I can propagate that fact and remove variables from the body of the rules

**Fact Extraction Algorithm**

**Input:** a KB K (Knowledge Base)

**Output:** all facts which must be true given K. All the variables that must be mapped to true

Algorithms that show the truth facts. Things that must be true given our knowledge

if I have X ← (X in necessary true)

If I have also a rule that use X in the mody I just remove X from all the body of the rules

Check again if there is another fact that has the variable X in the body (if there is one I remove X) and repeat until I can’t find any more variables to remove it from the body. This can be:

* because the body are all empty
* whichever rule has a non-empty body the variables that appear are NOT facts

**while** there is a fact x and a rule with x in the body **do** remove x from the body of all rules

**return** all facts in K

The KB (Knowledge Base) without facts in the body is called **redux**

At each interaction of the algorithm we remove all the facts from the body of all the rules, we get a simpler and simpler program all the time. The program that we obtain in the end before returning the facts is called **redux** of the Knowledge base

**Example**

Y ← X, Z, W

Z ← V, W

X ← U, V

Z ← W, U

V ← (it is a fact)

W ← (it is a fact)

Steps of the algorithm:

* Try to find a fact. It finds two facts

The fact V appears in the body of some rules

* go back to the program and remove the fact V from the body of all the rules that appears

Y ← X, Z, W

Z ← W

X ← U

Z ← W, U

W ←

from optimization also V← is removed

The fact W appears in the body of some rules

* go back to the program and remove the fact W from the body of all the rules that appears

Y ← X, Z

Z ←

X ← U

Z ← U

There is a new fact X, the X body is empty

remove X

Y ← Z

X ← U

Z ← U

DONE. We don’t have any more facts that appears in the body of some others rules

We have

* Y is true if Z is true
* X is true if U is true (we know that X is a fact but we can’t remove X ← U)
* Z is true if U is true

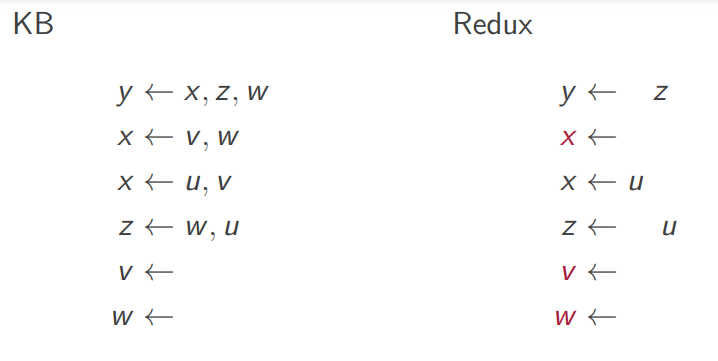
We don’t know if U is true

This is the **redux**:

Y ← Z

X ← U

Z ← U



**Correctness**

Unit resolution is **sound**

Need to make sure that this process is sound and complete

Whatever we conclude is a consequence is really a consequence and at the same time we derive all the consequences.

Soundness: most important one, we don’t want to conclude something that is false

Completeness: we would like it to, but sometimes we don’t achieve it

If V is a valuation such that

V (Y ← X1, …, Xi-1, Xi, Xi+1,, Xn) = 1 = V (Xi ← )

then V ( Y ← X1, … , Xi-1, Xi+i, … , Xn) = 1

Y ← X1, …, Xi-1, Xi, Xi+1,, Xn is the same as

Y v ¬ X1 v … ,v ¬ Xi, v ¬ Xn.

Doesn't matter the order with the disjunction, so is the same as

(Y v ¬ X1 v … v ¬ Xn) v ¬ Xi and add parentes it says: either Xi is false or this clauses (inside the partneses) is true

or (Y v ¬ X1 v … v ¬ Xn) is evaluated to 1 or Xi is evaluated to 1

Since we know that Xi is evaluated to 1 so ¬ Xi is evaluated to 0

(Y v ¬ X1 v … v ¬ Xn) must be evaluated to 1 it is the same as (Y ← X1, … Xi-1, Xi+1, … Xn) (without Xi)

This means that K → K^ is a tautology (*^* on top of K)

so the redux (symbol K^) is a consequence of the Knowledge (K)

In particular K → x is a tautology, if x is a fact in K^

Everything we derive from doing redux K^ is a consequence of the Knowledge K

**Correctness II**

The process is also complete w.r.t. (with respect to) facts

It return all the facts that are consequences of the knowledge base

if X is NOT a fact of K^ then it is not a consequence of K

The process extracts all (and only) the facts entailed by K

From the example

Y ← X, Z, W

Z ← V, W

X ← U, V

Z ← W, U

V ← (it is a fact)

W ← (it is a fact)

Y is not in the redux, we want to show that

K → Y is NOT a tautology (something that is always true in any possible evaluations)

To do so I need to find a valuation that satisfies my knowledge (K) but doesn't satisfy the variable Y. We look at the redux at at all the variables that becomes facts (X, Y, W) and make an evaluation that maps them to true, map all the other variables to false

U V W X Y Z

0 1 1 1 0 0

(1 for facts, 0 to no facts)

**Complex Entailments**

What if we are interested in a **complex** entailment?

K ⊨ Y ← X1, … Xn

The **relevant** valuations are those that satisfy K **and** all Xis



Add all the variables that appears in the premises and run the same algorithm

Y ← X, Z, W

Z ← V, W

X ← U, V

Z ← W, U

V ← (it is a fact)

W ← (it is a fact)

want to know K ⊨ Y ← U

want to know if whenever U is true Y is true

U is not a fact, add U as a fact in the KB (Knowledge Base)

Y ← X, Z, W

Z ← V, W

X ← U, V

Z ← W, U

V ←

W ←

U ←

Check if K’ ⊨ Y

process of unit of resolutions, remove the facts

Y ← X, Z, W

Z ← V, W

X ← U, V

Z ← W, U

V ←

W ←

U ←

we have

Y ←

**Computational Complexity**

The number of operations needed to find the consequences

How many computational resources we need to find an answer

How many steps we need to run our algorithm and get all the facts

We think about the worst case. In the worst case we must remove each variable from each rule. Do the most work possible.

Considering the worst case, this gives us our pessimist bound and we know that usually it takes less but then we know that it will not take more of that because we are considering the worst case

algorithm: look at the facts, if the facts appear in the body of some rules it will remove it.

Worst case: we take a variable and this variable appears in all the other rules, we need to go in all the rules to remove it. We find a new fact and it appears in all the rules again. Everytime we check a new fact we have to go to all the possible rules to remove it

If K has n rules and m variables, we need **n\*m** steps

10 rules and 5 variables, worst case is 50 steps

**Properties**

Rules define a simple KR language useful for expressing basic properties

fathers are male parents father ← male, parent

monometmata are egg-laying mammals monotreme ← mamman, oviparous

it it easy to extract consequences

**Limitations**

the language is very inexpressive

In particular, it cannot express relationship between objects

Can you define a grandparents, a sibling, an uncle? They depend on other objects.

To be a grandparent you need to be a parent, but your son has to be a parent, there are properties of others objects

We need some power from **predicate logic**