AI Notes

Viv Sedov — viv.sedov@hotmail.com

April 19, 2022

Contents

1 Propositional_logic

1.1 Simple Operations

When covering this : there are simple operations that you should know about :

Simple Operation Not:

p	$\neg p$
1	0
0	1

Such that in this case it is the opposite given value $\neg p$ would be the direct opposite of the given Value of P .

For example say that you have the following

p	q	$p \wedge q$
1	1	1
1	0	0
0	1	0
0	0	0

In this example, we are stating the following command : $\neg(14 > 6)$ is false , we create this truth table to prove if that is true or not .

$P \wedge Q$ is true $\iff$ p and q are true

Simple $\vee$ - OR

p	q	$p \vee q$
1	1	1
1	0	1
0	1	1
0	0	0

In this example above , for this to hold true, atleast one of the values would have to have a one it to hold true, this is known as an *Inclusive* or as in you can say the following and it would make sense: **I will go to the shops $\vee$ i will go to the coast**

Simple Operation $\implies$ This is where if something is true the other must be true , or where you given an equivalent pointer to if p then q

p	q	$p \implies q$
1	1	1
1	0	0
0	1	1
1	1	1

With the above example this is not cause and effect , there is a reason for why this occours , and that is that there is a pointer such that it acts like an if statment, with the following code shown below :

```
foo = True
if foo:
    return 1
else:
    return 0
```

The code above is rather simple , but shows that if something is true , then you would have some sort of value expression , or some pointer that would return if it is correct or not .

In the weird scenario of f and t , where if f is false it implies that q is true , that is because the q value is true , meaning that it would hold , a little trick for this one is that for what ever the second value is , if it is true , it will hold , if both are false , then it will true , though if one is true and the other is false , it will not hold .

Logically equivalent $\iff$ this can also be seen as if and only if or iff
Here is an example of the logical truth table behind this

p	q	$p \iff q$
1	1	1
1	0	0
0	1	0
0	0	1

With this , Where if something is true then the other must be true or if its false then the other would have to be false , in this case you are seeing if these two values are the same .

Example Exersises of how this would all work :
Given that :

$$p = Logicisfunforjaneq = daviddoesnotlikecabbager = davideyesareblue$$

We can then further express all of this in a notational form .

- $\neg p \wedge q$ This would imply the following truth table

$\neg p$	p	q	$\neg p \wedge q$
0	1	1	0
1	0	1	1
0	1	0	0
1	0	0	0

- $\neg R \wedge \neg P \implies$ Both are not goint to be true such that you would get only one possible answer for this .

Multiple truth table example : show the following in a truth table :

$$p \wedge (q \implies r)$$

p	q	r	$q \implies r$	$p \wedge (q \implies r)$
1	1	1	1	1
1	1	0	0	0
1	0	1	1	1
0	1	0	0	0
0	1	1	1	0
0	0	0	1	0

Here is another example to understand how these tables would all work together

$$(p \implies q) \wedge (q \implies p)$$

p	q	$q \implies p$	$p \implies q$	$(p \implies q) \wedge (q \implies p)$
1	1	1	1	1
0	1	0	1	0
1	0	1	0	1
0	0	1	1	1

Definition 1.1 Two propositions are equal if they have the same truth values , they are known as $P \implies Q$ this is known as logically equivalent

Definition 1.2 A proposition is tautology if it is always true an example of this is $P \vee \neg P$ this is always true

Definition 1.3 A proposition is a contradiction if it is always false for example $P \wedge \neg P$ this is always false

Definition 1.4 A proposition is **Contingent** if it is neither Always true or false

2 Disjunctive Normal Form

A given formula is said to be a *Disjunctive normal form* when it is an Or $\implies \vee$ this is known as **DNF** a function is conjunctive when it has an $\wedge$ form , within their proposition logic .

$$(p \wedge \neg q \wedge r) \vee (\neg q \wedge \neg r) \vee q$$

Everytime a given formula is built , we would folow the rules of propositional calculus , and how for each conjunctive formula there should be a disjunctive formula as well .

2.1 DNF of mini terms for truth tables

- For each row whose truth value is true , write doen for each of the prposition variables , of p_i in the formula of it self , either P_i is true in row or $\neg P_i$ if false.
- Repeate the first pointer , for the truth table where the formula is true and write down the dijunction of the conjuctions .

What you will rsee is that those two values will equal up such that the result of the formula in DNF is the equivalent to the original formula .

3 Conjunctive Normal Form

A formula is said to be **Conjunctive Normal Form** when its conjunction is $\wedge$ of disjunctive of $\vee$ an example of this is shown below :

$$(\neg P \vee Q \vee R \vee \neg S) \wedge (P \vee Q) \wedge \neg S \wedge (Q \vee \neg R \vee S)$$

Every expression built up according to the rules of calc , and suchj that for each conjunctive formula there is a similar or an equivelent formual that can be written in disjunctive form .

Conjunctive form , or in brackets , and on the outside
Disjunctive form, And in the brackets and or on the outside

4 Logical Equivalences

We can use this to obtain normal form , when we use the implication law to eliminate subproccess - when ever you have a doubel negation and demorgans to bring a ¬ you what this value to be on the outside : here are the sub process of how this can be done :

$$\neg\neg P \iff P$$

This rule is the double negation Law

$$\begin{aligned}(P \vee Q) &\iff (Q \vee P)\\ (P \wedge Q) &\iff (Q \wedge P)\\ (P \iff Q) &\iff (Q \iff P)\end{aligned}$$

Commutaive laws where both values would have to equal towards each other .

$$\begin{aligned}((P \vee Q) \vee R) &\iff (P \vee (Q \vee R))\\ ((P \wedge Q) \wedge R) &\iff (P \wedge (Q \wedge R))\end{aligned}$$

This is the associative laws , where it is very similar to how they work in matricies in which they can equate towards each other .

$$\begin{aligned}((P \vee Q) \wedge R) &\iff (P \vee (Q \wedge R))\\ ((P \wedge Q) \vee R) &\iff (P \wedge (Q \vee R))\end{aligned}$$

This lase is the distributive law , in which the given values would be changed within a DNF and CNF representation

$$\begin{aligned}(P \vee P) &\iff P\\ (P \wedge P) &\iff P\end{aligned}$$

Idempotent laws where the values of it self would always equal to it self no matter what .

Demorgans Law :

$$\begin{aligned}\neg(P \vee Q) &\iff (\neg P \wedge \neg Q)\\ \neg(P \wedge Q) &\iff (\neg P \vee \neg Q)\\ (P \wedge Q) &\iff \neg(\neg P \vee \neg Q)\\ (P \vee Q) &\iff \neg(\neg P \wedge \neg Q)\end{aligned}$$

Most times when you lookat demorgans law , you will notice that its very similar to the laws that have been stated above, but the thing that you want to note is that you will see that they are equal in some sense , where an or , is a direct link with Not and And it self.

§ Contrapositive Laws

$$(P \implies Q) \iff (\neg Q \implies \neg P)$$

implication that imply towards each other are contrapositive and hence you can switch out the given details of that infomation .

where If Q is an active reciever then P must be an active pointer is the same as stating if not p equates to Not q, in some sense you should understand how that would work .

§ Implication

$$(P \implies Q) \iff (\neg P \vee Q)$$
$$(P \implies Q) \iff \neg(P \wedge \neg Q)$$

§ Furhter implication

$$(P \vee Q) \iff (\neg P \implies Q)$$
$$(P \wedge Q) \iff \neg(\neg p \implies \neg Q)$$

Thi sone is rather annyoing , but the principle of how this works is very intriguing , if you do prove this via proof table you will see that they are truly equivalent: $P \vee Q$

p	q	$p \vee q$
1	1	1
1	0	1
0	1	1
0	0	0

this is the same as : $(\neg P \implies Q)$

P	Q	$p \implies Q$	$(\neg P \implies Q)$
1	1	1	1
0	1	1	1
1	0	0	1
0	0	1	0

If you look at the given tables above you will notice that indeed they are the same, a truth tabel may be long but they are very good at breaking down the given data that you have into something more readble .

Further Implies and equivalences

$$((P \implies R) \wedge (Q \implies R)) \iff ((P \vee Q) \implies R)$$
$$((P \implies Q) \wedge (P \implies R)) \iff ((P \implies (Q \wedge R))$$

With this law you are using the given equivelences that are shown above with the disjunctive and conjunctive views ,but within an equivelence ratio

§ Exportion Law

$$((P \wedge Q) \implies R) \iff (P \implies (Q \implies R))$$

This one is a good one , Mainly because if anything that does imply to another pointer , you can show that they are all equal towards each other .

§ Side Notes Within the compound proposition $\neg(P \vee Q)\&(\neg P \wedge \neg Q)$ they are the same , hence why when you look at the proof that is shown above you will see that they are the same .

When ever you look at equivelences you will notice that connectives $\vee\wedge$ will allways suggest that $P \vee Q \implies Q \vee P$

5 Formal definition

Definition 5.1,–,Valid An argument would be considered valid if and only if it takes a form that makes it impossible fro the premise to be true, in the sense that the conclusion is never going to be false. it is not possible to show it to be false in some sense. A formula is valid if and only if it is true under every understanding of its given argument, or its given schema, we can say it is valid if true holds for everything..

Definition 5.2,–,Sound implies Valid Valid allows us to imply Soundness

We can say that if you have a valid argument, then you also have a sound statement. am argument would be considered sound if it is valid and all the premise is true.

Definition 5.3 Sound, in logic a premise or conclusion is said to be valid if it is true under every possible understanding of its given argument, or its given schema.

1	2	1
24	5	1
7	8	1

The example their would be both valid and sound $\forall axiom \exists axiom \implies \vdash A$ What this just means is that A consist of either an axiom or can get derived from an axiom set using only the rules of inference

To Dumb this down even more, if you have statement x and you want to prove statement y, you can only do so by breaking x down into smaller statements to see if you can prove and show that y exists

$\forall x \exists y \implies \vdash y$
$\forall x \exists y \implies \vdash x \land y$
$\forall x \exists y \implies \vdash x \lor y$
$\forall x \exists y \implies \vdash x \implies y$
$\forall x \exists y \implies \vdash x \iff y$
$\forall x \exists y \implies \vdash x \leftarrow y$
$\forall x \exists y \implies \vdash x \rightarrow y$
$\forall x \exists y \implies \vdash x \leftrightarrow y$

Thing to note is that it does *Not* mean that **A is satisfied** this is a deduction but if you want to show satisfaction you would have to show

$A \ indicates \ \forall Axiom A$
$\forall A A(\models A) \implies True$

Theorem 5.1,–,Validity of statement Validity says nothing about whether or not any statement of the premises is true or not, it only says that the conclusion is true under every possible understanding of the premises. the key work their is Understanding of its own premises .

Such that validity states that its more about the form of an argument than it being true of it self.

So we can say a argument is valid if it has the proper form . An argument can have the right form, but be false.

```
Daffy Duck is a duck
All ducks are insects
their for daffy duck is a insect
```

Notice how these argument contain a form for *if x is Y* but then you see that they are not true. Notice However that if the premises **Were** True then the conclusion would also have to be true - this is a valid proof for validity. A valid argument needs not have true premises or a true conclusion.

Theorem 5.2,–,Sound requires a true premises Sound logically implies that a statement is true, this is due to the fact that, when a statement is sound, it means that it has a true premises and a true conclusion, we can formally derive x from y using this factor.

Soundness Is an argument or a factor if it means the following arguments

- It is valid
- it has a true premises

1. Sound requires both valid and to have a true premisies
2. for all valid arguments if the premises are true then the conlusion must also be true

Example :

1. All rabits are mammals
2. bugs bunny is a rabbit
3. Therfore, Bugs bunny is a mammal

In this argument we state that all of the premises are true, then the conclusion is true, so it is valid, and the premises are true, all rabbitss in fact are mammals and bugs bunny is a rabitt - so our conclusion makes sense.

Definition 5.4 Completeness

$$\alpha \models \beta \implies \alpha \vdash \beta$$

i.e if we can show something is true, then we can say that it is provable - we want to be able to prove all true statements, but you can also get false statements - such that you can prove both false and true Statements, such that if you end up proving false then your statement is no longer sound.

Definition 5.5 Soundness

$$\alpha \vdash \beta \implies \alpha \models \beta$$

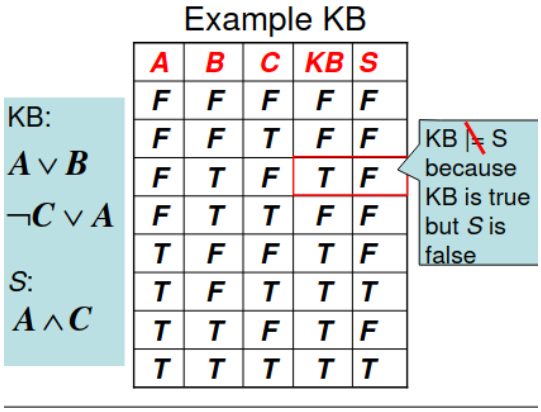
If we have a formulation i.e xy = 10, then we want to be able to show that fact. We do not want a system where we start out with something true and dedice something to be false, if we know something we should prove with our current knoweldge of breaking something down, that given statement would hold, through infierence rules.

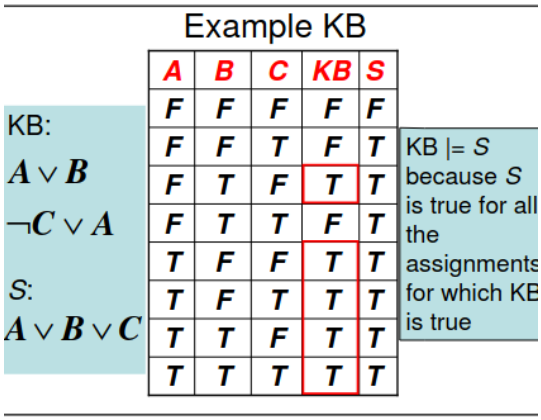
However it is conceivable that even if our system is sound, it maybe incomplete, regarding what it can express hence why it requires to have a competes property to ensure that our given formulation of our proof would hold true.

5.1 Summary without the crap

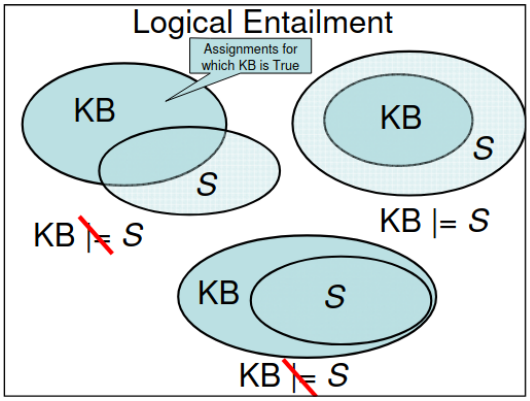
If your KB *Knowledge base* Entails Q then all interpretations (assigning true or false) values to variables that would allow you to evaluate your knowledge to True, also evaluates to Q to true $KB \models Q$

Entailment refers to how premises lead to a conclusion reacall how m(b) is a subset of m(a)





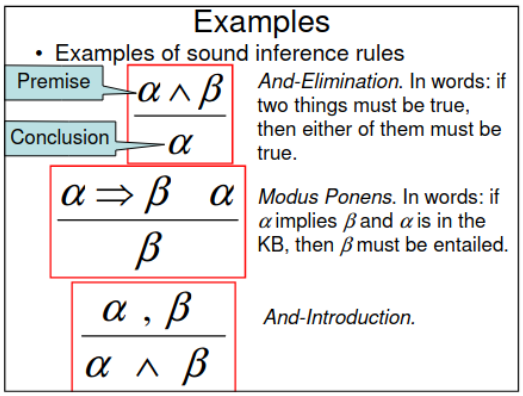
Leads into

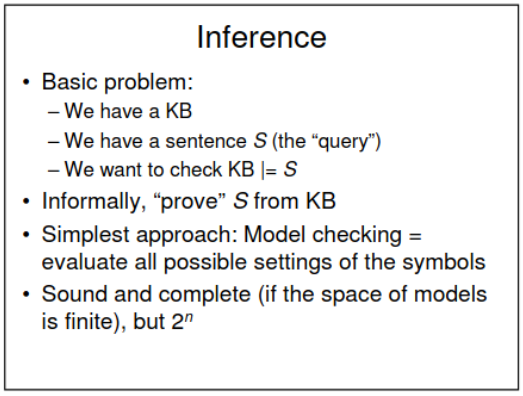


(1)

Inference is a procedure for deriving a new sentence 'p' from 'KB' following some algorithm. $KB \vdash p$ The inference algorithm is sound if it derives only sentences that are entailed by KB. The inference algorithm is complete if whatever can be entailed by KB can also be inferred from KB. Basically, an inference is an informal and less reliable kind of entailment.

- We have a kb
- We have some sentence S - query
- we want to prove S from our KB
- We say it is sound and complete if the space of model is finite within 2 pow n





5.2 Equivalences and normal form

Definition 5.6 A sentance is valid if it is true in all models

$$True, \alpha \vee \neg\alpha, \alpha \implies \alpha, (\alpha \wedge (\alpha \implies \beta)) \implies \beta$$

We can say that Validity is directctly connected through the inference of the deduction theorum

$$KB \models \alpha \iff (KB* \implies \alpha)$$

6 Resolution algorithm

- input Kb and S
- Output true if $KB \models S$ False otherwise
- Initalise a list of clauses CNF(KB and not S)
  - for each pair of clauses C i and c h

  - R implies resolution of i and j
  - new resolution is made
- If clauses are new then return false
- if cluauses unifie each other return true

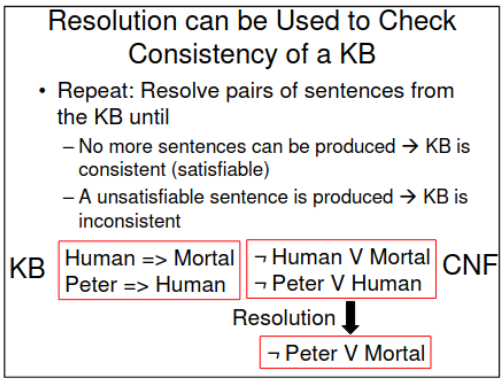
6.1 Resolution properties

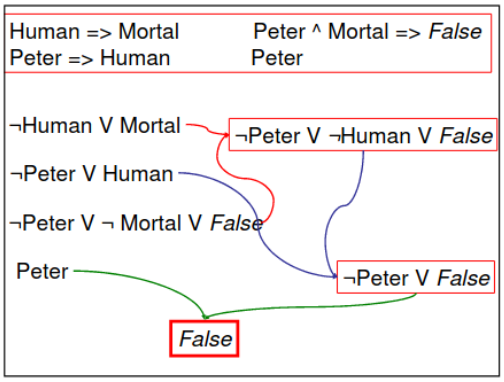
- Resolution is Sound i.e it produces a sentence that are entailed by their orginal owner
- Resolution is complete - it is guarantee to establish entailment of the query for every finite time
- Completeness is based on the key theorem

> **Theorem 6.1** If a set of clauses is unsatisfiable, then the set of all clauses that can be obtained by the resolution contains an empty clause

- So in short, we can say the opposite of a resolute theorem is that if we cannot find the empty clause the query must be satisfiable.

We use resolution to check consistency of how KB holds $Human \implies Mortal$
$Peter \implies Human$ We convert that into CNF $\neg Human \vee Mortal$
$\neg Peter \vee Human$
We can then give our resolution to this being $\neg Peter \vee Mortal$





(2)

Here in this image you can see how we can go from one proof, after another by linking them, to return if a valid argument will be true or false.

6.2 Horn Form KB

Horn Form is a clause or a form in which a logical inference problem is given : Given :

- A KB is a set of information
- A sentance $\alpha \implies theorem$

If a sentence is in KB, are restricted to some special form, some of the sound inference rules might be complete . In short its being able to convert from CNF form into implication rules
$(\alpha \vee \neg\beta) \wedge (\neg\alpha \vee \neg\gamma \vee \delta)$
$\iff$
$(\beta \implies \alpha) \wedge (\alpha \implies \gamma) \implies \delta)$ This is known as horn form normal form.

- Two inference rules that are sound and complete with respect to properistional symbols for kb in the hron normal form
  - resolution on a positive unit
  - Modus pones
- Have to be in conjuctive form $\vee \ldots \vee \ldots \vee \ldots$
- Would Follow the three basic principles
  - fact
  - Goal
  - Rule

  These are defined whith the following statement
- Rule : $ManandGel \implies Tall$ Saying Gel men are tall this can be counter proved with the following statement to prove completness $KB \models \alpha \iff (KB* \wedge \neg\alpha)$not provable to be satisfiable i.e unsatisfiab

  You want to end up showing that thing you are trying to prove may or may not have any models

7 Chaining

$\implies$ Just look at the slides

8 Total Summary of propositional logic

Theorem 8.1 Inference: process of deriving sentences entailed by the KB

$$KB \vdash_i \alpha$$

is a sentance where α can be derived from KB in teh process of i

For first-order logic there exists a sound and complete inference procedure - i.e. the procedure will answer any question whose answer follows from what is known by the KB.

8.1 Formal Definitions

Definition 8.1,–,Syntax *Syntax*: formal structure of how a sentence is made

Definition 8.2,–,Semantics *Symatics*: Truth of sentences with respect of the model

Definition 8.3,–,Entailment $KB \models \alpha \iff true \in \mathbf{T}, \forall Models \quad KB$

Definition 8.4,–,Sound Derivations produced only from entailed sentances

Definition 8.5,–,Completeness Can you prove your derived formulation from - any derivations can produce the same theorm in return

8.2 Syntax

Definition 8.6,–,Atomic $\bot$, $\neg \bot$ Or P $\implies$ Propositional logic for an atom

Definition 8.7,–,Negated Atomic $\neg$ anything with that just means its negated

Definition 8.8,–,Conjuction $\wedge$

Definition 8.9,–,Disjunction $\vee$

8.3 Equivalence Norms

Definition 8.10,–,Equivalence $\equiv$

Definition 8.11,–,Implication $\implies$

Definition 8.12,–,Negated Implication $\neg$

Definition 8.13,–,Biconditional $\iff$

- Conjuctive: Conjuction of disjunctions of literals

Definition 8.14,–,CNF $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$

- Disjunctive: Disjunction of conjuctions of literals

Definition 8.15,–,DNF $(A \wedge B) \vee (A \wedge \neg C) \vee (\neg A \wedge \neg D)$

8.4 Validity and Satisfiability wrt to KB

Theorem 8.2

$$KB \models \alpha \iff true \in \mathbf{T}Kb \models \alpha \iff (KB* \implies \alpha) \in \mathbf{T}$$

Note KB is a set of information in which $KB \implies \alpha$ Is not a wellformed formula - we get this through combining everything in the KB into one big conjuction.

8.4.1 CNTD Valid and Satisfiable

Definition 8.16 A sentence is satisfiable if it is true in some model

$$\alpha \vee \beta, \gamma$$

we can prove this by proving that α and β are true in some model and γ is true in some model we can derive values using modus pones

Definition 8.17,–,Unsatisfiable A sentence is unsatisfiable if it is false in some model

$$\alpha \wedge \neg\beta, \gamma$$

- Satisfiability is met only for the knolwedge base under the pretence that $KB \models \alpha \iff (Kb* \wedge \neg\alpha)$ which proving this will allow us to say this has no models and therefore is unsatisfiable

8.5 Modus Ponens

Definition 8.18,–,Modus Ponens $\forall\alpha \in KB \;\; \alpha \models \beta \;\; \beta \models \gamma \;\; \gamma \models \delta \;\; \delta \models \alpha$ Where
$\beta \in KB \;\; \gamma \in KB \;\; \delta \in KB \;\; \alpha \models \gamma \;\; \gamma \models \delta \;\; \delta \models \alpha$

8.6 Chaining

8.6.1 Forward Chaining

Forward chaining is the process of deriving a new sentence from a set of sentences that are already known.

Theorem 8.3

$$\begin{array}{l} \mathbf{KB} \vdash_i \alpha \\ \mathbf{KB} \vdash_i \beta \\ \mathbf{KB} \vdash_i \gamma \\ \mathbf{KB} \vdash_i \delta \\ \mathbf{KB} \vdash_i \epsilon \end{array} \Rightarrow \alpha \models \beta \models \gamma \models \delta \models \epsilon$$

- Forward chaining is a automatic process that can be used to derive new sentences from a set of known sentences.
- May do allot of dead work

8.6.2 Backward Chaining

Backward chaining is the process of deriving a new sentence from a set of sentences that are already known. the principle is teh same as forward chaining but we start from the end of the sentence and work our way back to the beginning.

8.7 Summary

Propositional Logic does not have enough power... We have this big kb and we have to go through each one to represent it. it is impracticle, forward and backward chaining are linear time, and are only *Complete* if you are working for horn clauses , else it is rather hard to convey.

9 First Order Logic

Note : Just use implication over and when you have

Definition 9.1 Constant : variables : functions

$$Knows(x, arithmatic)$$

'this is Atomic formula

$$Student(x) \implies knows(x, arithmetic)$$

'Connective, which is applies to a formula, Student knows arithmetic

$$\forall x Student(x) \implies Knows(x, arithmetic)$$

All students know arithmetic

Overall a great way of dealing with this, is actually making a graph, to convert the sentence into a graph like form, where you have different operations, and then check how they evaluate each other , in a sense this is a way how you convert propositional logic into first order logic.

Definition 9.2,–,Model in first order logic A model w in first order logic maps constant sysmbols to objects, similar to how connected graphs are all linked together
$w(alice) = \omega_1, w(bob) = \omega_2, w(arithmetic) = \omega_3$
Predicate symbols, to tuples of those objects $w(knows) = (\omega_1, \omega_3), (\omega_2, \omega_3), \ldots$

$\forall x \implies Y$ this is sound and works but if you have or use and with for all, then you are stating that everything and everything would be x and y
$\forall x \wedge y(x)$ see how that does not hold
$\exists xp \implies y$ this does not pair well
$\exists xP \wedge Y$ this is better

- And with Exist
- forall with Implies

9.1 Examples

- Which sentance best represents someones's mother is someone's female parent
  $\exists x, \exists y(Mother(x,y) \land (Female(x) \land Parent(x,y)))$
  Or
  $\forall x \forall y(Mother(x,y) \iff (Female(x) \land Parent(x,y)))$
  Or
  $\exists x \forall y(Mother(x,y) \implies (Female(x) \land Parent(x,y)))$ This can be quote opiniated, wrt containement, but lets read them through *Its the first and second one*

  - there exist some x and some y where there is a mother x ,y and that x is a female and that x y is a parent what that kinda means is saying that there can be a parent that is not a mother, in short you can say that y and x can be a parent but does not implcity imply that it can be a mother its a bit confusing but just go with that logic

9.1.1 Quantifiers order

The order between all given Quantifiers are not the same and can be swapped between the first order logic points

- $\forall x \forall y$ is the same as $\forall x \forall y \iff \forall y \forall x$
- $\exists x \exists y$ is the same as $\exists x \exists y \iff \exists y \exists x$
- $\forall x \exists y$ is *Not* the same as $\forall y \exists x \not\Longrightarrow \forall x \exists y$
- $\forall x \exists y Loves(x,y)$ Every x loves some y so *Everyone in the world is loved by atleast one person*
- $\exists x \forall y Loves(x,y)$Some x loves every Y is not the same as every x loves some y this is *There exist a person who loves everyone in the world* the reason for this is because when you have two $\forall A \forall B$ this is a pair, that you can sort through
- Universal Quantifiers: $\forall x p(x) \iff p(a) \land p(b) \land$
  $\exists x P(x) \iff p(a) \lor p(b) \lor$
- properties $\neg \forall x P(x) \iff \exists x \neg P(x)$
  Demorgans law with or applies within logic
- Usefull to convert natural language sentences to first order Sentences :
  - Every student knows arithmetic
  - We can use $\forall$ $\forall x student(x) \implies knows(x, arithmetic)$ But if you use and, then you are saying, something else, or you are implying something else.

9.1.2 Quantifier duality

Duality is important as these quantifiers are reversable So you can say something like this

$\forall x\ likes(x, icecream) \equiv \neg\exists x\ \neg likes(x, icecram) \qquad \exists x\ Likes(c, brocolai) \equiv \neg\forall x\ \neg likes(x, brocolai)$

9.2 More Examples to work on

- $\forall x, y(brother(x, y) \implies Sibling(x, y))$
  This is saying, for every x, if brother(x,y) then that logiclly implies every brother is a sibling
  $\forall xy(Sibling(x, y) \iff Sibling(y, x))$
  Every x there is a y where x and y and y and x will always be a sibling

- A first cousin, is a child of a parents sibling
  $\forall x, y(FirstCoursin(x, y) \implies \exists p, p' Parent(p, x) \land Sibling(P', p) \land Parent(P', x))$
  for every x y , you have a first cousin under the condition that x and y is a firstcoursin for x to y and that there exist some p and p prime where parent of x is P and sibling of P prime is p and parent of y is P prime

9.2.1 Wumpus world example

- Squares are breezy near a it : *Diagnostic rule*, this is where infierence would come into play, where you infer *Cause from effect* $\forall y Breeezy(y) \implies \exists x$ Pit$(x) \land Adjacent(x, y)$
- Casual Rule : infern effect from cause $\forall x, y$ Pit $(x) \land Adjacent(x, y) \implies Breezy(y)$
- None of this is complete, the point is we done imply each other , though this can be built to become complete $\forall y Breezy(y) \iff [\exists xPit(x) \land Adjacent(x, y)]$

10 Usefull notes from the quiz

- Information about the knowledge base
  - If a model is true in the *Real world* then any sentance derived from that given model is sound through inference, procedure, and hence is entailed within the real world
- How do you tell if an agent is true in the real world ?
  - If the agent has sensors, that allow you to create the connection with the sentance and that the agents knoledge base is sound then it would be true within the real world
  - The ai agents learning ability generates general rules from expierence that the ai belives to be true, this is can be faliable, as this depends on how good the agent is at expressing its own information
- A satisfiable sentance is a sentance that is true in all models : *True*
- Following statements are unstatisfiable $(\alpha \land \beta) \land (\neg\alpha \land (\neg\alpha \implies \neg\beta))$ this will always resolve to zero
  $\Delta \land \neg\Delta$ This will also result in zero, anything with contains a negative will never be satisifable
- Horn Clause examples

- $(\neg B_1 \vee \neg B_2 \ldots \neg B_n \vee C)$Horn clause rule, everything is negative but one
- B This is true
- $\neg(D_1 \vee d_2 \vee d_3)$Everything is a negative -¿ Goal state
- Proof facts
  - to prove $KB \models \alpha$ is equivalent to showing that $Kb* \wedge \neg\alpha$ has no models
  - the expression $Kb* \implies \alpha$ and $KB \models \alpha$ are connected through deduction theorem

- ***Problems with Propositional**logic* $\implies$ ***does not support variables*** Propositinal logic has an issue where it does not support variables, due to this, your statement would have to be very large

  this is true, propositional logic works through the factor that you work with values, and cant assign anything directicly as it propositional logic follows two main principles

  - Declarative, pieces of syntax respond to facts
  - Compositional meaning that $p_{12} \wedge B_{11}$ is derived from me meaning of p over 1 2 and b over 1 1

10.1 Examples

There is some course that every student has taken $\exists y Course(y) \wedge [\forall x Student(x) \implies Takes(x, y)]$ Remember exist contains and, and then for all have a implies

How about

Every even integer greater than 2 is the sum of two primes $\forall x even(x) \wedge greater(x, 2) \implies \exists y \exists z Same(x, sum(z, y)) \wedge prime(y) \wedge (prime(z))$ Sometimes you have two primes, you can state that you are using two prims and they are EE values

If a student takes a course and the course covers a concept then the student knows that concept

- if =¿ universal quantification means you have a bunch of them

10.2 First order logic proofs

All human lectures are happy $\forall x(Lecture(x) \wedge Human(x) \implies Happy(x))$

- clauses are a disjunctive form of literals that hae to be converted in most cases
- A definite clause is a clause where everything is negative but one
- A definite clause tends to be when you have not(..... and .... and ....) or x Then you can use implication to show that exists for the values that you are working with .

Susan bought everything that tony bought $(\forall x Bought(Tony, x) \implies Bought(Suzan, x))$

10.2.1 Inferences in First Order Logic (proofs)

How do you prove through your kB:

Definition 10.1,–,Subsitution First order logic contains unification and Substitution, this is due to the fact that this logic principle contains Variables, similar to how Prolog would work $\delta = \frac{x}{John}$ Given a sentence S and a substitution
$S = King(x)$
$\delta = \frac{x}{John}$
$S\delta = King(John)$

Definition 10.2,–,Universal instance Every instance that we make when we unify some value, to let a given value poses over another in some sense Unification is the process of making a substitution, and this is done by unifying the two values . They use each other.

$\frac{\forall x \alpha}{\alpha 0}$ For any variable v and ground term g if 0 is the subsitution of v/g then we can say the about

Here you replacing values, v through g, so you are allowed to replace variables, think of how python lets you change the variable name

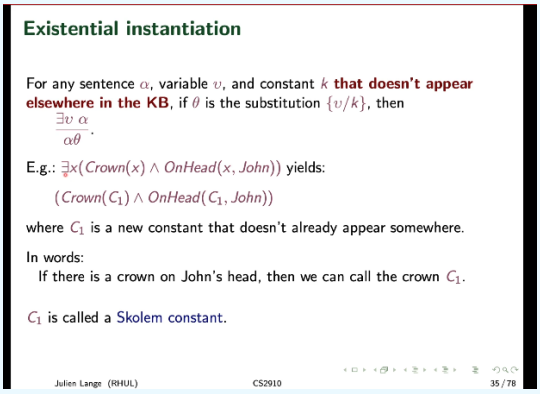
In this example, what you are doing is replacing x, as you are replacing x with john, richard etc, if you know something about all possible vs, then you can use one

Definition 10.3,–,ground term A ground term is a term that contains no variables

$\forall x(King(a) \wedge Greedy(x) \implies Evil(x))$yields$\forall x(king(John) \wedge Greedy(John) \implies Evil(John))$

Ground terms are easy to work with.

Definition 10.4,–,Existential Instantiation For any sentence α and v and constant K that does not need to apear anywhere but the KB, then we can say the following



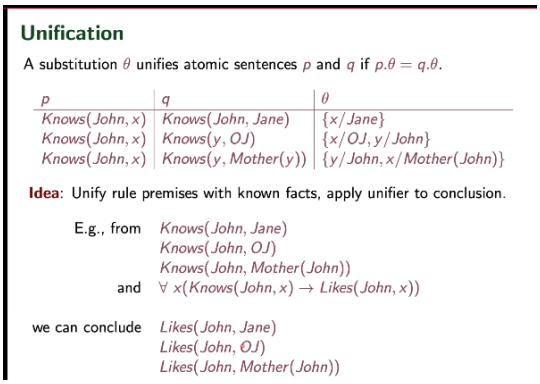
(3)

C1 needs to be brand new and make sure that, it does not appear anywhere within the state, because we know that it would have to be on the head of the king, pretty much .

- If you have universal instantiation , then *We can instantiate if we have an value from the qunatified formula - so we can just replace x with some name as a ground term*
- If you have universal Instantiation , then *We can say, we can replace the quantifiable Value with a new Name, so we can just get rid of it and give it a big Capital Boi to name it, shown in the image above*

10.2.2 Unification

How do you replace variables, Just look at the prolog notes i guess, as the principles would be the same



(4)

- Say you know that you are trying to find two people, here we can replace John , x with x = Jane Its a unification, where you replace x with jane, and then both formulas would become the same.
- For the second case, X needs to become Oj and Y would have to become John when you are replacing values

Where tf did you get Likes from

Definition 10.5,–,General Unifier These are the unifiers that do not have any or add any constraints, when you want an ai system towards the best answers, and quickly, so you want the answer to be as simple and quickly as possible.

So we want our unifiers to be as simple as possible

10.2.3 Unification Algo

Given any two formulas $\alpha\beta$ there is a very fast way of unifying things together

$UNIFY(\alpha, \beta) = 0 \iff \alpha 0 = \beta 0$ Which checks whether alpha and betra can be unified, with the the least or general unifier within our db for example if you have
$p(x, y, f(z))$
$p(f(y), A, x)$
we can unify the following
$\frac{x}{f(a)}$
$\frac{y}{A}$

$\frac{z}{A}$

Unification algorithm (informal algorithm)

Given $\alpha = f(\alpha_1, \ldots, \alpha_k)$ $\beta = g(\beta_1, \ldots, \beta_n)$, we want to unify α and β.

1. Check whether $f = g$ and $k = n$: if not, **fail**; otherwise, continue:
2. Build a temporary substitution: $\theta = \{\alpha_1/\beta_1, \ldots, \alpha_k/\alpha_n\}$
3. Repeat the following till no transformation applies (and stop with success)
   1. Select any pair of the form x/x and **remove** it
   2. Select any pair of the form t/x and **rewrite** it to x/t
   3. Select any pair of the form x/t and **apply** it on θ (i.e., replace x by t in every right-hand side)
   4. Select any pair of the form $f'(\ldots)/g'(\ldots)$ and **unify** them (i.e., apply the algorithm recursively). Add output to θ.

(5)

- Unification is a "pattern matching" procedure that takes two atomic sentences, called literals, as input, and returns "failure" if they do not match and a substitution list, Theta, if they do match. That is, unify(p,q) = Theta means subst(Theta, p) = subst(Theta, q) for two atomic sentences p and q.
- Theta is called the most general unifier (mgu)
- All variables in the given two literals are implicitly universally quantified
- To make literals match, replace (universally-quantified) variables by terms

```
procedure unify(p, q, theta)
  Scan p and q left-to-right and find the first corresponding
     terms where p and q "disagree"  ; where p and q not equal
  If there is no disagreement, return theta  ; success
  Let r and s be the terms in p and q, respectively,
     where disagreement first occurs
  If variable(r) then
     theta = union(theta, {r/s})
     unify(subst(theta, p), subst(theta, q), theta)
  else if variable(s) then
     theta = union(theta, {s/r})
     unify(subst(theta, p), subst(theta, q), theta)
  else return "failure"
end
```

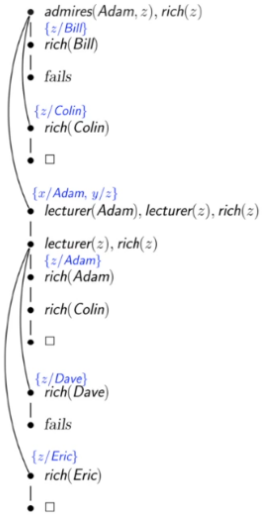
10.2.4 Forward Chaining Proof

We can use forward chaining to prove that this factor holds

- Sound and complete fore first order definite clauses
- may not terminate in general if α is not entailed
- You need entailment with definite clauses
- *DataLog* = First Order Logic definite Clauses + no functions They are an iterative functional value that you work of with .

10.3 First Order Logic End Summery

Say we have the following sentence Which Rich person Does Admam admire ?
$admire(Admin, Bill)$
$admire(Admin, Colin)$
$admire(x, y) \leftarrow lectures(x), lectures(y)$
$lecturer(Adam)$
$lectureer(Dav)$
$lectureer(Eric)$
$rich(colin)$
$rich(Eric)$
$rich(Adam) \leftarrow rich(colin)$
If we do backtracking we can do the following to find our solution



The computation backtracks whenever a 'dead end' is reached ('*fails*') or when a solution is found (□).
The curved arcs show the backtracking points and the subsequent computation.
The answers to the query will therefore be produced in the following order:

$z = Colin$
$z = Adam$
$z = Eric$

(6)

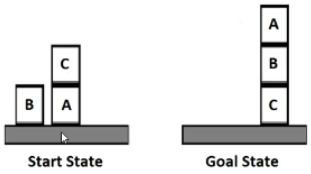
This is a nicer way of showing a proof to a given kb

11 Temporal Reasoning

11.1 Classification of Ai temporal reasoning problems

- Prediction problem : i.e project problem, given an inital state and casual rule, describing a domain, we want to derive the state of the world resulting from some given sequence of actions Think of how a chess game would work, thiunk of a stack, when you are going through data .
- Planning problem : Planning problem given a description of the inital state, how are you going to change to rules to get to your domain state, through what actions will allow you to get to that point ?

- The planning problem – given a description of the initial state and some causal rules describing a domain ("domain description") we want to derive a sequence of actions (or some other structure of actions) that will lead from the initial state to a specified goal state.



(7)

- Explanation problem - given some casual rules describing a domain, we want to discover facts about our given state *You wake up in the morning and you head downstairs in the kitchen theres a plate on the table and a bowl with a little milk left in, you can then say by assumption that your housemake was awoke before you and already had their breakfast*

11.2 Terminology

Definition 11.1,–,Fluents Where the Truth varies over a period of time, think of them as a boolean state variable, Like when a person x lies, but sometimes they can state the truth, it would follow that logic

we can also call this : *Temporal Propositional statements*

Definition 11.2,–,Time instances and time periods In short, year month day, or what happend on that date, we state to be time instances for example Last year my university decided to do in person exams to F every CS student over this allows us to give an initiation of our time which leads into what occoured

We generate new fluences when a state is changed, for example in the image above, when you have your start state, b [c,[a]] in case you see that every variable there is in flux, whihch would imply that we would have to move c from a, and then move b to a etc ... to get to our goal state

11.3 Frame problem

The frame problem is about finding a great way to handle non change .

- Peristence or Intertia - the assumption that the facts are not affect by its actions - such that they would hold the same truth value after the action, as they had before hadn
- The root of the problem lies in the fact of what the domain would represent
- For smaller domains, with just a small number of fluents, and action types, we can just write it out properly.

To most AI researchers, the frame problem is the challenge of representing the effects of action in logic without having to represent explicitly a large number of intuitively obvious non-effects types of problems

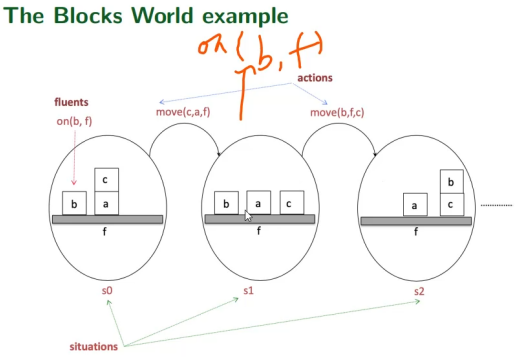
1. Qualification problem - exhuastivly specifying preconditions of an action
2. Computational fram to verify what was inferred
3. Ramification problem - problem of exhastively specifying the effects of an action - think of chess

TODO: Write more about Frame problem, as im not 100 % sure how this works

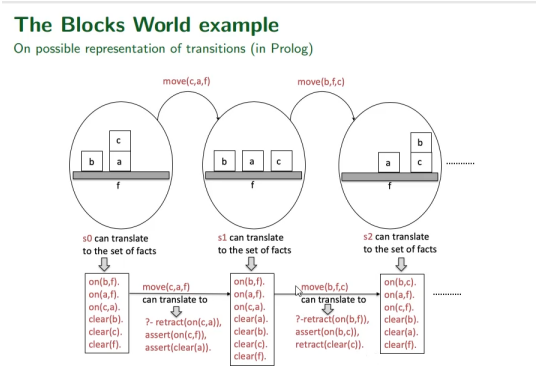
11.4 Situational Calculus

The situation calculus is a logic formalism designed for representing and reasoning about dynamical domains. It was first introduced by John McCarthy in 1963. The main version of the situational calculus that is presented in this article is based on that introduced by Ray Reiter in 1991. Wikipedia *Domain - tend to be in first order logic :*

- Fluents bassicly the item you are trying to change
- Actions the function you are pushing on those fluents
- Situations like an instance of the current state



(8)



(9)

- Remover and append operatiions within a fleucen
- Representation of actions as it stands does not check whether it is possible for a block to be moved from one place to another $\implies$ They dont check if something has been moved
- How can you improve cause and effect

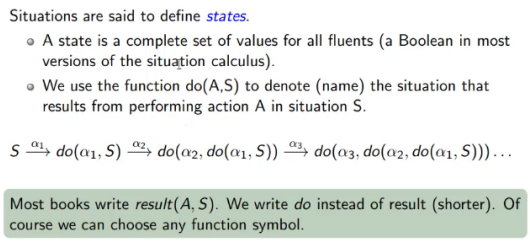
Situation Calculus is trying to deal with the reasoning of the present

We can use situational calculus to represent the current condition of a state.

11.4.1 Situational Calc state Representation

Situational Calc is defined through states

if you look at the image above we have multiple situations, from s1 to s3, with different fluents and actions that have been made upon some given variable



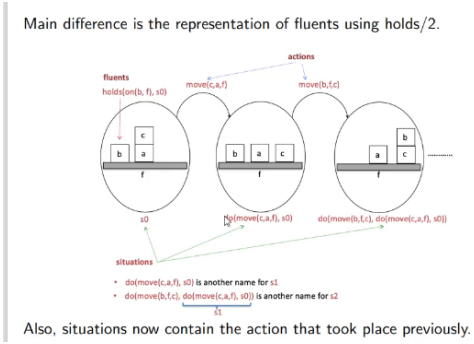
(10)

If you initial state was S, and then you have this functional operation, a_1 then you would evaluate to a function that would be $do(a_1, S)$, and this is a recursive functino or a recursive call that is made. These Do values here are new states, are when an action state, where they would be a new situation after previous situation, in which that would lead to another state.

11.4.2 Language

Situational Calc, is a bassis that relies of the following language

- Constant and variables : *These are for fluents and actions* they are usally sorted .
- names For situations, basic functional values, where we have some function that would denote some action that is being done within that given time
- *Holds(F,S)* Represents a fluent F is true in the situation S
  What this means is when ever something i shold, it means it will contain to be true within that state .



(11)

Main difference is that representation of fluents using hold is that we base an action, on the new solution ? ? ?? wtf is this ?

Move(current$_s tate||Whatisiton||Wheredoesitgo$)

Here you would use a hold, to check if something is true or not, we use hold to show a fluent . Which would mean that b on top of f is true in s0, both are valid way of showing fluents

- ***Main Idea***

  - What holes true after an action occurs
  - What remains the same after an action occurs
  - What changes after an action
  - What were the pre conditions for that action to take place - and was it derived from previous actions.

Definition 11.3,–,Language of situational calculus Difference between situation and state

- A state is the set of fluents that are always true
- A Situation provides a history of transitions (actions) from initial state So think of the recursive idea, where you held all the previous states in one go, its similar to that principle .

11.4.3 Axioms

Axioms are statements that are always true, and are used to define the language of the situation calculus. They are preconditions that are specified when working with an action

1. Preconditions effect - Axioms A fluent p holds in the situation do (, s) so long as P and execution of alpha in S makes P true, and the precondition of A if any are all true for S. So in some sense, when we have a function, we have to make sure that it would hold true, on execution such that it makes sense

Consider the process of specifying actions.

- **Precondition-Effect axioms**
  A fluent P holds in situation $do(\alpha, S)$ if P and execution of α in S makes P true and the preconditions of α (if any) are true in S.

```
% X is on Z after we move it from Y to Z.
holds(on(X, Z), do(move(X, Y, Z), S)) ←
        holds(on(X, Y), S),
        holds(clear(X), S),
        holds(clear(Z), S).

% Y is clear after we move X from it (to somewhere else i.e. Z).
holds(clear(Y), do(move(X, Y, Z), S)) ←
        holds(on(X, Y), S),
        holds(clear(X), S),
        holds(clear(Z), S).
```

(12)

So in this image it looks confusing and your right, its way too much crap for what its worth So in short, when we say x is on z after we move y to z

action(move y to z) means (x is on z)

So when we have on(x,z) we have to show, what actions were made, so move(x,y,z) right, the arror denotes the preconditions, to the movement of the value. Such that x had to be on y so we can move from y, and then we have to clear x and clear z i.e we have to check that nothing was on top of x and y

These are knowns as preconditions being created.

2. Frame Axioms Indicating A fluent P holds in situation, where S is not stop, i.a you have a continuous axiom of representation.

   Basically When something does not change, and your action state is doing something else, and not effecting your current clear or on value that you are working with.

12 Inductive Learning

Inductive Learning is pretty much, decision trees, high level, symbolic approach to how properties would work . ***Legit if else statements : think of disjunctive normal form***

Definition 12.1,–,Learning Learn is defined through the following items that allow us to measure how much an item can learn, a good approach to think about is, maybe how much IQ a person has ?

1. E = Experience
2. T = Task
3. P = Performance

The learning is measured such that Our experience is based with respect to our task and that how well have we learning, such that we base that off our performance.

12.1 Why is learning Key to this ?

- System is based on agents within the real world, all be it sensors, or agents that we have
- some cases we have to have decision making to improve our performance
- For environments that may not be fully known, some cases we have to work within respect to what the agent would learn
  - Acting on its own making its own rules through pattern matching
  - Handle data from our agent and much more

As always learning, is a basis that allow us to increase our performance over time, learning improves performance, because it can learn through its conditions, goals and other items within our system or our KB

12.2 Forms of Learning

There are different forms of learning

12.2.1 Unsupervised Learning

Definition 12.2,–,Unsupervised Learning Learn through patterns, within our data set. : There is no external application in which you can go out your way to help and say oh you can do this .
You have zero feedback and you learn with respect to the input data, over a period of time.
Examples :

- Self Learning Cars; trying to figure out what is good traffic or not.

12.2.2 Reinforcement Learning

Definition 12.3,–,Reinforcement Learning this is where you learn from series of rewards and punishments, think of a monkey, whenthey learn, or when a baby learns, there are expierences that they have to go through to get to where they are now, and there is a similar pricniple to what happens here

you have the Benefits and the downsides, which need to be in balance but only learn through telling the agent this is what oyu can do, this is what you cannot do:

think of it like, when you are trying to teach someone to drive somewhere.

12.2.3 Supervised Learning

Definition 12.4,–,Supervised Learning Agent observes some examples, you give it a training set and it learns from that training set: good example is when you messed around with neural networks to lean about this basis right, recall when you messed around with the mnist dataset, you had to give it a network and self learning properties but first you had to give it a data set for it to work with right. exact same principle here

Inductive Learning

Definition 12.5,–,Inductive Learning Inductive learning is like human learning based on past experiences

- the closest way of an ai agent to learn is to do so from observations or giving it the information or telling it to do something with respect to its previous data
- Agent will use learning of a target function to predict the values of an attribute, in which you can approve or not, or have certain qualifiers, where you can say the following thing about the given time or information you have you induce information: i.e inductive
  - Approve
  - not approved
  - high risk
  - low risk

Induction: Is a way to show how we can compare to things to gather, and say here are two sets of information, deduce some information from it, so we can work with new information, you are creating new info in some sense.

This is a variant of supervised learning

So with supervised learning we have these tasks of inference: *What data do we have to learn from*

- Training data
- test set
- Some Idea of what we are looking for

How does Inductive Learning work

- Inductive learning is the basis for Choosing the best hypothesis, i.e the best data that we can work with
- our h Would have to be consistent, or that we know that there are no issues with our data set
- ***Learning problem realisable*** If $G(h) \in H$ then we can say the following
  - H = is a set of all valid functions that we can work with, i.e hypothesis or the expressive value that we got
  - if our G(consist value h) is within our hypothesis than we say our Learning problem is realisable