Logic for computer science.

"self reference problem".

Logic (or anything) has to be studied using Logic.

* we need Logic; to predict the behaviour of computational object.

So, learn Logic.

Logic: infer conclusions, from given premises.

- propositional logic] deals with propositions.
- First order Logic.
  - doesn't look inside the proposition.

Eg: pen is cheap.
and
pen is gold.

looks inside the proposition.

- parametrized propositions
- quantifiers.

1) Godel's incompleteness:-

"there are theories whoes assumptions cannot be listed."

number theory.

* consider a list of axioms.

"this statement cannot be proven by the list".

Age of Philosophy:-

- Syntax:-

set Vars.

$Vars = \{P_1, P_2, \dots\}$

countable many propositional variables.

* $\top$ true (paranthesis
$\bot$ false.)
$\neg$ negation
⋮

* propositional formula F is defined as:- (Generation rules).

if $F \in P$ (the set of propositional formulas.)
then
$\top \in P$
$\bot \in P$, vars⊂P
$\neg F \in P$
$(F_1 \circ F_2) \in P$.

P is the smallest set of propositional formulas.

* Vars(F) is the set of variables appearing in F.

!

* do following strings belong to P?

p ✓
$\neg p$ ✓
(p) X
$p \Rightarrow q$ X
$(p \Rightarrow q)$ ✓

} paranthesis is a strict rule! GRAMMAR

(¬p) X
¬¬¬p ✓

* $\top$, $\bot$, $p(\in Vars)$ are called atomic formulae. (↷ undividable.)

$\neg p$ is not atomic.

→ Parse tree:-

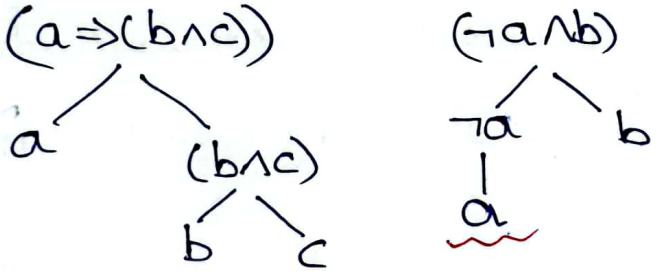
Thm:- $F \in P$ iff F is obtained by unfolding of generation rules.

* parse tree of F is a tree, such that

i) root is F.

ii) Leaves are atomic formulae.

iii) each internal node, is formed by formation rule on its children.



Thm: $F \in P \iff F$ has a parse tree.

Thm: $F \in P \implies F$ has a unique parse tree

NOTE:-

parsing produces a DAG, not tree..

all are different

* $(P_1 \wedge (\neg P_1 \vee P_2))$

$(\neg P_1 \vee P_2)$

$\neg P_1$ P_2

P_1

* Subformula:-

G is subformula of F; if G occurs within F.

G is proper subformula of F; if $G \neq F$ & is a subformula.

* Nodes of parse tree are subformulas.
root✓ internal✓ leaf✓.

* Sub(F) is the set of all subformulas of F.

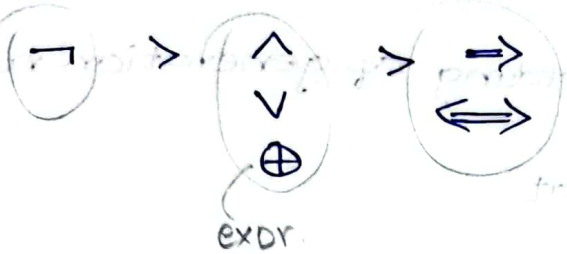
∴ elemVars of (F) also subformula of F.

* immediate subformula: children in parse tree.

* Leading connective: root of two child nodes.

→ Shorthand:-

introduce precedence.
(to omit paranthasis)

* 

* $F_0 O_1 F_1 O_2 F_2 \cdots O_n F_n$, F_i is either atomic,

or Bracketclosed and (or) negation.

negation isn't atomic!

i) find O_i such that preference *precedance $O_i > O_{i+1}$

reduction $n \to n-1$

* $p \vee q \vee r$ means?

$p \Rightarrow q \Rightarrow r$ means?

Associativity preference. (left or right).

→ most prefer right.

$p \Rightarrow q \Rightarrow r$ means $(p \Rightarrow (q \Rightarrow r))$

→ Substitution:-

F is a formula.

* $F[G_1/p_1, G_2/p_2 \ldots G_n/p_n]$ is simultaneous substitution.

* $F[G_1/p_1][G_2/p_2]$ — is sequential...
put G_i in place of p_i in F.

* Eg: $F(p,q) = \neg p \oplus q \Rightarrow h$

then $F(r \vee q, T) = \neg(r \vee q) \oplus T \Rightarrow h$

* countability:-
(cardinality)

Defn:- $|A| \leq |B|$ if there is a one-one function $f: A \to B$.
even for ∞ sets.

countable:-
A set A is said to be countable, if $|A| \leq |N|$
natural nos.

if $|A| > |N|$ then uncountable.
⇒ NO one-one function from A to N.

* if there is a onto, one-one fun; then $|A| = |B|$.

* if "Symbs" is countable, Symbs* is also countable.
Alphabet; Words

let f: Symbs $\to N$.

& p_i be ith prime. i can be arbitraly large.

we define $h(a_1, a_2 \ldots a_k) = \prod_{i=1}^{k} p_i^{f(a_i)}$

∴ the primes are used as co-ordinates

∴ h: Symbs* $\to N$ is one-one.

Lecture 3.

Semantics:- Assigning meaning.

Till now; we never said; but now we are saying any proposition will have values **0** (or) **1**.

(our innate intuition is
0 → false
1 → true)

→ MODEL:-

A **model** is an element of $\{ Vars \longrightarrow \mathbb{B} \}$

set of inputs.

"configuration" of inputs.

boolean $\mathbb{B} \triangleq \{0,1\}$

i.e.
⇒ an assignment of values to every element of Vars.
0 or 1.

∴ $\{ P_1 \longrightarrow 1, P_2 \longrightarrow 0, P_3 \longrightarrow 0, \dots \}$ is a model.

* A model **m** may or may not satisfy a formula F.

satisfaction relation is shown as $m \models F$ in infix notation

↳ again a proposition.

→ $m \models F$; satisfaction relation:-

the satisfaction relation $\models$ between models & Formulas is the smallest relation;

$m \models T$

$m \models P$ if $m(P) = 1$ (P: element of Vars.)

$m \not\models 1$ ↳ never satisfies false.

$m \models F$ if $m \not\models \neg F$ (F: Formula. (Not 'false'))

$m \models F_1 \vee F_2$ if $m \models F_1$ or $m \models F_2$

$m \models F_1 \wedge F_2$ if $m \models F_1$ and $m \models F_2$

$m \models F_1 \Leftrightarrow F_2$ if $m \models F_1$ iff $m \models F_2$

* F is satisfiable if there is some model m; $m \models F$.

F is Valid/Tautology if for each model m; $m \models F$. $\models F$ — notation

F is unsatisfiable if for each model m; $m \not\models F$. $\not\models F$

→ Overloading $\models$:

i) Set of models on LHS:-

let M be a set of models

$M \models F \iff \forall m \in M,\ m \models F$.

(natural)

ii) Set of formulas on LHS:-

let Σ be (possibly infinite) set of Formulas.

$\Sigma \models F \iff$ for each model m satisfying all formulas in Σ;

$m \models F$.

Σ implies F.

* meaning; $\Sigma_1 \wedge \Sigma_2 \cdots \wedge \Sigma_n \Rightarrow F$

(elements of Σ)

a) equivalent:-

$F \equiv G$ iff; for each model m

$m \models F \iff m \models G$. (both have same truth table).

Eg: $p \vee (q \vee r) \equiv (p \vee q) \vee r$.

b) equi satisfiable:-

F & G are equisatisfiable if (need not be equivalent.)

F is Sat $\iff$ G is Sat

need not be, for the same model m.

Sat → satisfiable.

SAT solver → satisfiability solver.

① equivalid:-

F, G are equivalid if

$\models F \iff FG$

notation of "is valid".

* "equisatisfiable" is used in formula transformation.

Tseitin's encoding?

F, F ∨ G are equisatisfiable, if F is sat.

Decidable:-

A problem is decidable, if there exists an algo to solve. whether be P or NP.

Satisfiability prblm:-

For a given F ∈ P, is F satisfiable?

this we can "decide" after writing down truthtable.

∴ decidable problem.

but Algo is exponential: 2^(inputs) truthtable.

* validity problem:-

for a given F ∈ P, is F valid?

Valid ≠ satisfiable.

valid = tautology.

* $p \oplus q = (\neg p \wedge q) \vee (p \wedge \neg q) = \neg(p \iff q)$

the expressive power; formula.

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

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

→ Boolean function:-

* A finite boolean function is $f: \mathbb{B}^n \to \mathbb{B}$

Eg: $P_1 \vee P_2$ represents $f = \{(0,0) \to 0, (1,0) \to 1, (0,1) \to 1, (1,1) \to 1\}$.

expressive power:-

for every finite boolean function f; there is a formula F that represents f.

Eg: with $\neg, \vee, \wedge$; we can construct F.

* Minimal logical connectives:-

$\{\neg, \vee\}$ can define all other logical connectives.

$\{\neg, \wedge\}$ ✓

$\{\wedge\}$ ×

Q: prove $\{\neg, \oplus\}$ is not fully expressionable...

How?

Age of mathematics. (Formal proofs)

*Say for a set of formulas Σ; & a formula F; $\Sigma \vdash F$

"Sigma implies F".

we need to syntactically infer this; without truth tables; since Σ "could" contain ∞ Formulae.

we (derive) (by using):-

$\Sigma \vdash F$

"sigma proves F"

proof is our main aim.

→ Proof rules:-

means to derive new statements, from old statements.

RuleName $\frac{\text{old proofs}}{\text{new stmt.}}$ Conditions

1) Assumption $\frac{}{\Sigma \vdash F}$ $F \in \Sigma$

2) Monotonic $\frac{\Sigma \vdash F}{\Sigma' \vdash F}$ $\Sigma \subseteq \Sigma'$

3) DoubleNeg $\frac{\Sigma \vdash F}{\Sigma \vdash \neg\neg F}$

these are rules!

→ Derivation:-

A list of statements; derived from old statements.

Eg: 1) $\{p \vee q, \neg\neg q\} \vdash \neg\neg q$ — Assumption proof rule.

2) $\{p \vee q, \neg\neg q, s\} \vdash \neg\neg q$ — monotonic proof rule, applied to 1. (premise.)

this process is derivation.

! don't question them.

→ Proof rules for $\wedge$:-

Proof rule for $\neg$ is double neg.

$\wedge$-INTRO $\dfrac{\Sigma \vdash F \quad \Sigma \vdash G}{\Sigma \vdash F \wedge G}$

$\wedge$-ELIM $\dfrac{\Sigma \vdash F \wedge G}{\Sigma \vdash F}$ } this is rule! so:

$\wedge$-ELIM $\dfrac{\Sigma \vdash F \wedge G}{\Sigma \vdash G}$ ✗

$\wedge$-ELIM $\dfrac{\Sigma \vdash G \wedge F}{\Sigma \vdash G}$ ✓ strict

$\wedge$-SYMM $\dfrac{\Sigma \vdash F \wedge G}{\Sigma \vdash G \wedge F}$

$\wedge$-PAREN $\dfrac{\Sigma \vdash (F \wedge G) \wedge H}{\Sigma \vdash F \wedge G \wedge H}$

→ For $\vee$:-

$\vee$-INTRO $\dfrac{\Sigma \vdash F}{\Sigma \vdash F \vee G}$

$\vee$-PAREN $\dfrac{\Sigma \vdash (F \vee G) \vee H}{\Sigma \vdash F \vee G \vee H}$

$\vee$-SYMM $\dfrac{\Sigma \vdash F \vee G}{\Sigma \vdash G \vee F}$

$\vee$-DEF $\dfrac{\Sigma \vdash F \vee G}{\Sigma \vdash \neg(\neg F \wedge \neg G)}$ definition.

$\vee$-DEF $\dfrac{\Sigma \vdash \neg(\neg F \wedge \neg G)}{\Sigma \vdash F \vee G}$ } but no $\wedge$-DEF bcoz;

essentially de-morgan rule.

we "take" $\neg, \wedge$ to be base.

$\vee$-ELIM $\dfrac{\Sigma \vdash F \vee G \quad \Sigma \cup \{F\} \vdash H \quad \Sigma \cup \{G\} \vdash H}{\Sigma \vdash H}$ } very useful.

Since $\Sigma \vdash F \vee \neg F$.

∴ $\vee$-definition is in terms of $\neg, \wedge$. ✓

→ for $\Rightarrow$:-

$\Rightarrow$-INTRO $\dfrac{\Sigma \cup \{F\} \vdash G}{\Sigma \vdash (F \Rightarrow G)}$

Nothing like $\Rightarrow$-SYMM

$\Rightarrow$-ELIM $\dfrac{\Sigma \vdash F \Rightarrow G, \Sigma \vdash F}{\Sigma \vdash G}$

$\Rightarrow$-DEF $\dfrac{\Sigma \vdash F \Rightarrow G}{\Sigma \vdash \neg F \vee G}$ definition.

$\Rightarrow$-DEF $\dfrac{\Sigma \vdash \neg F \vee G}{\Sigma \vdash F \Rightarrow G}$

Q.) prove $\phi \vdash (p \Rightarrow q) \vee p$ · using proof rules.

indirectly ; show $\phi \vdash p \vee \neg p$.

Sol)

1. $\{p\} \vdash p$	Assumption.
2. $\phi \vdash p \Rightarrow p$	$\Rightarrow$- INTRO to 1
3. $\phi \vdash \neg p \vee p$	$\Rightarrow$- DEF to 2
4. $\phi \vdash p \vee \neg p$	V- SYMM to 3

Proof rules for ( ), $\Leftrightarrow$, $\oplus$:-

$\Leftrightarrow$DEF $\dfrac{\Sigma \vdash F \Leftrightarrow G}{\Sigma \vdash F \Rightarrow G}$; $\Leftrightarrow$-DEF $\dfrac{\Sigma \vdash F \Leftrightarrow G}{\Sigma \vdash G \Rightarrow F}$; $\Leftrightarrow$-DEF $\dfrac{\Sigma \vdash F \Rightarrow G, \Sigma \vdash G \Rightarrow F}{\Sigma \vdash F \Leftrightarrow G}$

4.3. SOUNDNESS :- of proof.

Theorem :-

if $\Sigma \vdash F$, then $\Sigma \models F$.

($\vdash$: proves ; $\models$: implies)

(stating only one direction)

Proof : by case analysis... & induction. — 1. Take true for premise.
2. Show for proved statements.

enough if we prove the above theorem for "Proof rules" since our derivation is, in steps of proof rules.

---

* The before discussed proof rules are not redundant!

i.e. none can be derived from the remaining others.

* if we wrote reverse doubleNeg; along with doubleNeg; then it will become redundant.

5.1 Derived rules.

→

* 1) Modus ponens:- (saying; if F implies G & F is true; then G is true.)

$\Sigma \vdash \neg F \vee G$
$\Sigma \vdash F$ then $\Sigma \vdash G$.

Proof:

1) $\Sigma \vdash \neg F \vee G$, premise
2) $\Sigma \vdash F \Rightarrow G$ ⇒-DEF. to 1
3) $\Sigma \vdash F$ premise.
4) $\Sigma \vdash G$ ⇒-ELIM to 2,3.

Eg:-

1. $\Sigma \vdash q$ premise.
2. $\Sigma \vdash R \wedge p$ premise.
3. $\Sigma \vdash \neg q \vee p$ premise
4. $\Sigma \vdash p$ V-modus ponens to 1,3

---

2) Tautology:-

any set Σ; $\Sigma \vdash \neg F \vee F$!

---

3) Contradiction:-

if we have $\Sigma \vdash F \wedge \neg F$, then $\Sigma \vdash G$

$F \wedge \neg F$ is false na... so write absurdity in conclusion.

literally anything

Proof: (101)

1. $\Sigma \vdash F \wedge \neg F$ premise
2. $\Sigma \vdash \neg F \wedge F$ ∧-SYMM to 1.
3. $\Sigma \vdash \neg F$ ∧-ELIM to 2.
4. $\Sigma \vdash \neg F \vee G$ V-INTRO 3.
5. $\Sigma \vdash F$ ∧-ELIM to 1.
6. $\Sigma \vdash G$ V-MODUS PONENS to 4. & 5.

4. Contrapositive:-

if $\Sigma \cup \{F\} \vdash G$ then $\Sigma \cup \{\neg G\} \vdash \neg F$.

$$\text{CONTRAPOSITIVE}\ \frac{\Sigma \cup \{F\} \vdash G}{\Sigma \cup \{\neg G\} \vdash \neg F}$$

More proof methods:-

Proof by cases:-

$\Sigma \cup \{F\} \vdash G$
$\Sigma \cup \{\neg F\} \vdash G$ then $\Sigma \vdash G$

Proof:-

1. $\Sigma \cup \{F\} \vdash G$ premise
2. $\Sigma \cup \{\neg F\} \vdash G$ premise.
3. $\Sigma \vdash F \vee \neg F$ tautology.
4. $\Sigma \vdash G$ $\vee$-Elim.

$$\text{ByCases}\ \frac{\Sigma \cup \{F\} \vdash G \quad \Sigma \cup \{\neg F\} \vdash G}{\Sigma \vdash G}$$

Proof by contradiction:-

$\Sigma \cup \{F\} \vdash G$
$\Sigma \cup \{F\} \vdash \neg G$ then $\Sigma \vdash \neg F$ sweet!

Proof:-

1. $\Sigma \cup \{F\} \vdash G$ premise
2. $\Sigma \cup \{\neg G\} \vdash \neg F$ contrapositive 1.
3. $\Sigma \cup \{F\} \vdash \neg G$ premise.
4. $\Sigma \cup \{\neg\neg G\} \vdash \neg F$ contra 2.
5. $\Sigma \vdash \neg F$ ByCases 1 3,4

Reverse double Negation:-

if $\Sigma \vdash \neg\neg F$ then $\Sigma \vdash F$. didn't write it before, bcoz our proof rules will become REDUNDANT

Proof:-

1. $\Sigma \vdash \neg\neg F$ premise.
2. $\Sigma \cup \{\neg F\} \vdash \neg\neg F$. Monotonic 1.
3. $\Sigma \cup \{\neg F\} \vdash \neg F$ assumption.
4. $\Sigma \cup \{\neg F\} \vdash \neg F \wedge \neg\neg F$ $\wedge$-Intro 2,3
5. $\Sigma \cup \{\neg F\} \vdash F$ contradict^n 4.
6. $\Sigma \cup \{F\} \vdash F$ Assumption
7. $\Sigma \vdash F$ ByCases 5,6.

$$\text{REVDOUBLENEG}\ \frac{\Sigma \vdash \neg\neg F}{\Sigma \vdash F}$$

★ pmp!

→ **Resolution:-** very Imp. Criterion.

if we have $\Sigma \vdash \neg F \vee G$ and $\Sigma \vdash F \vee H$, then $\Sigma \vdash G \vee H$

generalisation of modusponens.

Proof:

1. $\Sigma \vdash \neg F \vee G$
2. $\Sigma \cup \{F\} \vdash (\neg F \vee G)$ monotonic to 1
3. $\Sigma \cup \{F\} \vdash F$ Assumption.
4. $\Sigma \cup \{F\} \vdash (F \Rightarrow G)$ $\Rightarrow$-DEF to 2.
5. $\Sigma \cup \{F\} \vdash G$ $\Rightarrow$-ELIM to 4, 3
6. $\Sigma \cup \{F\} \vdash G \vee H$ V-INTRO to 5.

(F is true, then $G \vee H$ is proved.)

7. $\Sigma \vdash F \vee H$

$\Sigma \cup \{\neg F\} \vdash F \vee H$.

$\Sigma \cup \{\neg F\} \vdash \neg F$.

Show: $\Sigma \vdash \neg F \vee H$ Now...

we got to show.

$\Sigma \cup \{\neg F\} \vdash G \vee H$

finally....

RESOLUTION $\dfrac{\Sigma \vdash \neg F \vee G \quad \Sigma \vdash F \vee H}{\Sigma \vdash G \vee H}$

this is THE proof rule.

1. $\Sigma \vdash F \vee H$ premise.
2. $\Sigma \cup \{F\} \vdash F$ Assumption.
3. $\Sigma \cup \{F\} \vdash \neg\neg F$ DOUBLENEG to (2)
4. $\Sigma \cup \{F\} \vdash \neg\neg F \vee H$ V-Intro to (3)
5. $\Sigma \cup \{H\} \vdash H$ Assumption.
6. $\Sigma \cup \{H\} \vdash H \vee \neg\neg F$ V-Intro to 4.
7. $\Sigma \cup \{H\} \vdash \neg\neg F \vee H$ V-Symm to 6.
8. $\Sigma \vdash \neg\neg F \vee H$ V-elim to 1, 4, 7.

Being a good student rewards!

★ imp..

Tada!

Such a tedious process!

just to flip sign.

"one hand tied to back"

Substitution?:- won't use this as a proof rule.

(Like; ¬¬F in place of F. (FVG) in place of (¬F∧¬G) Like these!

Not:- (F∨¬G) in place of R.

Theorem 5.9:-

Let $F_1(p)$ and $F_2(p)$ be two formulas.

we have $\Sigma \vdash F_1(G) \Leftrightarrow F_1(H)$

$\Sigma \vdash F_2(G) \Leftrightarrow F_2(H)$

and $\Sigma \vdash F_1(G) \wedge F_2(G)$

$G \equiv H$ here.

then $\Sigma \vdash F_1(H) \wedge F_2(H)$ (Straight forward proof. use $\Leftrightarrow$-DEF). & $\wedge$-ELIM.

Similar results for $\vee, \neg, \oplus, \Rightarrow, \Leftrightarrow$ (whole propositional logic.) ✓

& Hence; a more wholesome result will be...

Theorem 5.10:-

Let F(p) be a formula.

if $\Sigma \vdash G \Leftrightarrow H$

& $\Sigma \vdash F(G)$

then; we can drive → amalgamation of all individual steps.

$\Sigma \vdash F(H)$.

(But we won't use substitution as proof rule).

"A proof is a proof, if it is easy to check".

- Ashutosh sir.

* Substitution will be a linear time cost.

Our proof system should (preferably), be constant time cost.

Lecture 6:-

Principle of Structural induction:-

Every formula in P has a property Q if:-

Base Case:- every atomic formula has property Q.

$(\top, \bot, p)$

$\neg p$ is not atomic.

Induction step:-

if $F, G \in P$ have property Q,

so do $\neg F, (F \circ G)$ where $\circ$ is a binary symbol.

Theorem 6.2:- "Substitution theorem".

let $F(p), G, H$ be formulas & m be a model

if $\left[m \models G \text{ iff } m \models H\right]$ then $\left[m \models F(G) \text{ iff } m \models F(H)\right]$.

meaning

$m \models G$ and $m \models H$ fine

$m \not\models G$ and $m \not\models H$ fine fine!!

use structural induction.

Theorem 6.3:- "Equivalence Generalisation theorem"

if $F(p) \equiv G(p)$ then for each formula H, $F(H) \equiv G(H)$

Theorem 6.4:- "Subformula replacement theorem"

let $G, H, F(p)$ be formulas & if $G \equiv H$ then $F(G) \equiv F(H)$.

Associativity:-

$\vee, \wedge, \oplus$ are associative.

∴ we don't need paranthesis in following formula

$p_1 \wedge p_2 \wedge p_3 \wedge \cdots p_n$

$p_1 \vee p_2 \vee p_3 \vee \cdots p_n$

$p_1 \oplus p_2 \oplus p_3 \oplus \cdots p_n$

Flattening : drop of paranthesis.

Extended Distributivity of $\vee$ over $\wedge$. (for CNF flattening).

$$\bigvee_{i=0}^{m} \bigwedge_{j=0}^{n_i} p_{ij} \equiv \bigwedge_{j_0=0}^{n_0} \cdots \bigwedge_{j_m=0}^{n_m} \bigvee_{i=0}^{m} p_{i j_i}$$ (proof by induction).

$$\left(p_{00} \wedge p_{01} \wedge p_{02} \cdots \wedge p_{0 n_0} \right) \vee \left(p_{10} \wedge p_{11} \cdots \wedge p_{1 n_1} \right) \vee \cdots$$

$=$

Explosion.

$(1+n_0)(1+n_1)(1+n_2)(\cdots)(1+n_m)$ terms finally.

tseitin's encoding saves the day.

Age of Computer Science:-

Normal forms:-

1) Negation normal form.
2) Conjunctive normal form.

* $p \Rightarrow q \equiv (\neg p \vee q)$

$p \oplus q \equiv (p \vee q) \wedge (\neg p \vee \neg q)$

$p \Leftrightarrow q \equiv \neg(p \oplus q)$

NNF definition:-

A formula is NNF if $\neg$ appears only in front of atoms.

$a \wedge b \equiv \neg(\neg a \vee \neg b)$

NNF not NNF.

* Terminology:-

proposition variables → atoms.

Literal → atom or its negation.

clause → disjunction of literals

CNF is a set of clauses.

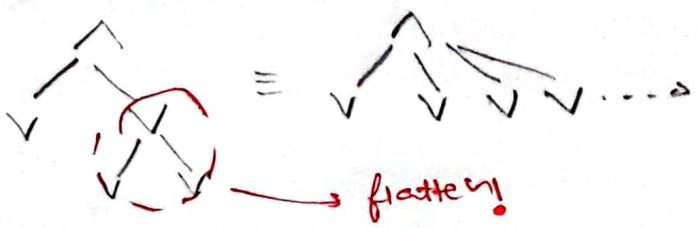
conjuction of clauses.

* $\neg p$, p, $(p \vee \neg q)$ are all in CNF.

* every formula can be equivalence to a CNF formula.

* Proof:

1. remove $\oplus$, $\Rightarrow$, $\Leftrightarrow$ using equivalences.
2. convert the formula into NNF.
3. Flatten the $\vee$ and $\wedge$.



* we will bring

(∧) above the (∨) symbol

by distribution.

(this explodes the size! So, we will see tseitin's coding).

$p \vee \neg p \vee p \vee q$ is a clause.

$\{p, \neg p, q\}$ is the same clause.

// CNF.

→ Tseitin's encoding:-

consider $p \vee (q \wedge r)$. (not CNF).

* we replace $(q \wedge r)$ by x & add clauses to encode that x is almost like $(q \wedge r)$

P.T.O. first

$$(p \vee x) \wedge (x \Rightarrow (q \wedge r))$$

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

Note that $p \vee (q \wedge r)$ & $(p \vee x) \wedge (\neg x \vee q) \wedge (\neg x \vee r)$ are NOT EQUIVALENT. They are equisatisfiable.

NO exponential explosion.

linear growth.

$$(x_1 \wedge x_2 \wedge \dots x_m) \vee (y_1 \wedge y_2 \dots y_n)$$

$$\equiv$$

$$(h \vee k) \wedge (\neg h \vee x_1) \wedge \dots (\neg h \vee x_m) \wedge$$

tseitin's encoding $\wedge\ (\neg k \vee y_1) \wedge \dots (\neg k \vee y_n)$

$(1 + m + n) << (m \cdot n)$

Note that NNF/CNF form is necessary for tseitin to work.

Problem:

$$(a_1 \wedge b_1) \vee (a_2 \wedge b_2) \vee (a_3 \wedge b_3).$$

convert into tseitin encoding!

(we learnt tseitin only for two terms... what to do?)

$$(a_1 \wedge b_1) \vee \Big((a_2 \wedge b_2) \vee (a_3 \wedge b_3) \Big)$$

$$= (\ ) \vee \Big((x \vee y) \wedge (\neg x \vee a_2) \wedge (\neg x \vee b_2) \wedge (\neg y \vee a_3) \wedge (\neg y \vee b_3) \Big)$$

wrong! the new terms should be separate.

Defn:-

1. Assume input formula is NNF without $\oplus, \Rightarrow, \Leftrightarrow$.
2. Find a $G_1 \wedge G_2 \wedge \cdots G_n$ that is below an $\vee$ in $F(G_1 \wedge G_2 \wedge \cdots G_n)$ (need not innermost, just to serve our cases)
3. Replace $F(G_1 \wedge G_2 \wedge \cdots G_n)$ by

$$F(p) \wedge (\neg p \vee G_1) \wedge \cdots (\neg p \vee G_n)$$

$p \rightarrow$ fresh variable.

So;

now, the new terms are conjucted with whole $F(p)$.

$$\therefore = \Big[(\ ) \vee (x \vee y) \Big] \wedge (\neg x \vee a_2) \wedge (\neg x \vee b_2) \wedge (\neg y \vee a_3) \wedge (\neg y \vee b_3)$$

$F(p)$

but above; you wrote in a subformula

$$= (z \vee x \vee y) \wedge (\quad) \wedge (\neg z \vee a_1) \wedge (\neg z \vee b_1)$$

(1) (4) (2)

total 7 terms.

→ Solving a few families of SAT problems:-

1) K-Sat :-

A K-Sat formula is a CNF ϕ has atmost K-literals in a clause.

Eg:- $(p \wedge q, \wedge r)$ is 1-SAT.

$(p \vee q \vee r)$ is 3-SAT

Theorem:- K-sat to 3-SAT equisatisfiability.

For each K-SAT formula F; there is a 3-SAT formula F' with linear blowup such that F,F' are equisatisfiable.

(linear blowup: phew! otherwise useless.)

(doesn't mean that same model must gotta satisfy both).

Proof:-

let a clause be $(l_1 \vee l_2 \vee l_3 \cdots l_k)$. in formula F.

then

$G' = (l_1 \vee l_2 \vee x_2) \wedge (\neg x_2 \vee x_3 \vee l_3) \wedge \cdots (\neg x_{K-3} \vee x_{K-2} \vee l_{K-2}) \wedge (\neg x_{K-2} \vee l_{K-1} \vee l_K)$

(intelligent!)

is an equisatisfiable 3-SAT.

where $x_2, x_3, \cdots x_{K-2}$ are ∉ Vars(F).

∴ F is sat ⟺ $F - \{(l_1 \vee l_2 \cdots \vee l_k)\} \cup G'$ is sat.

∴ K became 2K-1 atoms;

or rather;

K literals became 3K literals.

3x size at worst.

linear!

* General 3-SAT is NP-complete. Cook-Levin theorem.

But 2-SAT is polynomial. (we use implication graphs)

* we will discuss polynomial time subclasses of SAT problem.

the most general "satisfiability".

1) 2-SAT (implication graph)
2) XOR-SAT (elimination)
3) Horn clauses. (from AI & ML). (all null to a normal model).

note to the curious:- Schaefer's Dichotomy theorem identifies subclasses of SAT problem that are polynomial; and all others are NP-complete. The above three subclasses play an important role in the theorem.

The theorem suggests that no other subclass is polynomial.

→ 2-SAT:- Implication graphs.

2-SAT is CNF; that has only binary clauses.

(p is (p∨p)).

repeate unary clauses!

Eg: $(p \vee q) \wedge (\neg p \vee p) \wedge (\neg p \vee r)$ is a 2-SAT.

→ Implication graph:-

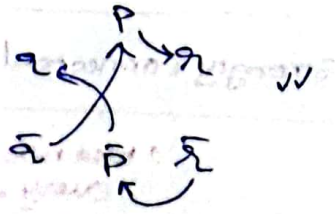
let F be a 2-SAT; & $Vars(F) = \{p_1, p_2 \dots p_n\}$.

Implication graph (V,E) as

$V = \{p_1\ p_2 \dots p_n\ \neg p_1\ \neg p_2 \dots \neg p_n\}$

$E \triangleq \{(\bar{l}_1, l_2), (\bar{l}_2, l_1) \mid (l_1 \vee l_2) \in F\}$

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



• properties of implication graph:-

1. if there is $l_1 \to l_2$; then there is, $\bar{l_2} \to \bar{l_1}$.

2. if there is a SSC - $S \subseteq V$; then there's a complementary SSC $S^c \subseteq V$; that has literals exact negation.

3. for each m ∈ F; if $l_1 \to l_2$ is there in graph;
   then (if $m(l_1) = 1$ then $m(l_2) = 1$)
   but if $m(l_1) = 0$ then dunno!.

4. in a SSC;
   if $m(l_1) = 1$; $m(SSC) = 1$
   $m(l_1) = 0$; $m(SSC) = 0$
   bcoz;
   for any two l_1, l_2.
   $l_1 \to l_2$ ($\because \bar{l_2} \to \bar{l_1}$)
   & $l_2 \to l_1$ too! (since SSC).!.
   ↓ not trivial.
   $\bar{l_2} \to \bar{l_1}$ is trivial.

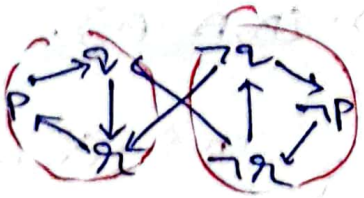
→ reduced implication graph:-

reduced implication DAG (V^R, E^R) is defined as;

$V^R \triangleq \{S \mid S \text{ is a SSC in } (V, E)\}$

$E^R \triangleq \{(S_1, S_2) \mid \text{there are } l_1 \in S_1, l_2 \in S_2 \text{ s.t. } (l_1, l_2) \in E\}$

Eg:



implication graph.

reduced DAG.

strongly connected ≠ complete graph

need not have every edge. just connectivity...

has every edge.

2-SAT satisfiability:-

Theorem 8.7

(also gives a rule for SAT)

A 2-SAT formula is unsat iff there is a SSC S in its implication graph (V,E) such that $\{p, \neg p\} \subseteq S$ for some p.

hmmm...

Proof:-

reverse:

$\because p \rightarrow \neg p \;\&\; \neg p \rightarrow p$

absurd! no sat!

forward:- in a contrapositive manner.

Assume that there is no such S.

solving: we will construct a sat-model of F as follows:-

1) Initially, all literals are unassigned.

2) while {some SSC in (V,E) is unassigned}

- let $S \in V^R$ be an unassigned SSC whose all children are 1.
- Assign literals of S to 1. So, S^c is 0.

$\{p, \neg q, \neg r\}$ ✓✓

not vertices.
but yes literals!

↓

while-loop. ✓

* we will never find a unassigned node with child marked as 0, because if

complement space.

but; i wont ever assign 1 to a node which has children unassigned na!.

Hecky!

* ~~polynomial~~ even better; linear! time;

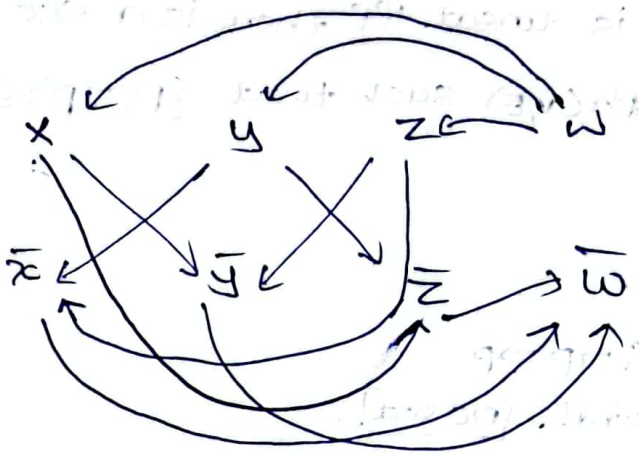
due to above proof.

Q) solve.

$$(\neg x \vee \neg y) \wedge (\neg y \vee \neg z) \wedge (\neg z \vee \neg x) \wedge (x \vee \neg w) \wedge (y \vee \neg w) \wedge (z \vee \neg w)$$

Sol)

implication graph:-



$\bar{w} \rightarrow 1$

then, $\left.\begin{matrix} \bar{x} \\ \bar{y} \\ \bar{z} \end{matrix}\right\} = 1.$

$\therefore \{0,0,0,0\}$ is a model ✓.

→ XOR-SAT:- (Need not be 2-SAT. This is for K-SAT)

A XOR-SAT formula is a conjuction of XORs of literals.

Eg: $(p \oplus q \oplus r) \wedge (q \oplus \neg r)$ ✓.

* $p \oplus q$ is $\neg(p \Leftrightarrow q)$. That is; xor is negation of equal.

$(p \Leftrightarrow \neg q)$

∴ if $p \oplus q$ is true; we say p equals $\neg q$.

"p not equals q"

Theorem:

for variable p; formula G,F; if $p \notin Vars(G)$ then

$(p \oplus G) \wedge F$ and $F[\neg G/p]$ are equisatisfiable.

Sure!

Eg: $(p \oplus r \oplus s) \wedge (q \oplus r \oplus s) \wedge (\neg p \oplus q \oplus \neg s) \wedge (p \oplus \neg q \oplus \neg r)$

① • $P = \neg(r \oplus s) \equiv \neg r \oplus s$.

can't distribute XOR over OR, AND.

So

$(q \oplus r \oplus s) \wedge ((r \oplus s) \oplus q \oplus \neg s) \wedge (\neg r \oplus s \oplus \neg q \oplus \neg r)$

Gaussian Elimination

$(q \oplus r \oplus s) \wedge (1 \oplus s \oplus q) \wedge (0 \oplus s \oplus \neg q)$

$\equiv \neg s$ careful!

$\equiv s$!

② $s = q$

$(q \oplus r \oplus q) \wedge (\neg q \oplus q) \wedge (q \oplus \neg q)$

$q \oplus q = 0$! ① ①

$\Rightarrow (r) \wedge (T) \wedge (T)$

Be wary! ***

$a \oplus a = 0$

$a \oplus \neg a = 1$

$0 \oplus a = a$

$1 \oplus a = \neg a$

∴ $r = 1$. $q = 0$, $s = 0$; then $p \equiv 0 \oplus 0 = 0$.

{0,0,1,0} is one model!.

HORN CLAUSES:-

is a clause; that has the form:

$(\neg p_1 \vee \neg p_2 \vee \cdots \neg p_n \vee q)$ } same as $(p_1 \wedge p_2 \wedge \cdots p_n \Rightarrow q)$

where $p_1, p_2, \cdots p_n \in$ Vars

$q \in$ Vars $\cup \{\bot\}$

I think AI/ML will have lots of these! yeaa...

* the clauses with $q \in$ Vars are implication clauses.

with $q \triangleq \bot$ are goal clauses.

Eg:

$\{ p, \neg q \vee \neg r, \neg p \vee q, \neg r, \neg t \vee \bot, \neg p \vee \neg q \vee \neg t \vee \bot \}$

$T \Rightarrow p$ implication clause. goal clause. goal clause.

How to solve?

Algo HornSat(Hs, Gs)

input: Hs: implication clauses Gs: Goal clauses.

output: model / unsat

```
> m := λx.0; (whole model is 0)
> while m ⊭ (p1 ∧ ... pn ⇒ p) ∈ Hs do
>     m := m[p → 1];
> if m ⊭ (q1 ∧ q2 ∧ ... qk ⇒ ⊥) ∈ Gs then return "UNSAT";
> return m;
```

Not linear time, I think. even though model updating is linear changes.

"recognizable Horn SAT problems"

maybe we rename $a \vee b$ as $\neg p_1 \vee b$.

Lecture-9:- Resolution proof system:

refutation proof system → try to prove UNSAT.

(even for $\Sigma \vdash G$; we do $\Sigma \cup \{\neg G\}$ is UNSAT lol).

* we can treat CNF as a set of clauses.

∴ Σ ⟷ CNF is intuitive.

Also a set of formula ::: clauses.

→ Best thing: this proof system; only 2 rules.

1. ASSUMPTION $\dfrac{}{\Sigma \vdash C}$ $C \in \Sigma$

2. RESOLUTION $\dfrac{\Sigma \vdash F \vee G \quad \Sigma \vdash \neg F \vee H}{\Sigma \vdash G \vee H}$

if we have multiple pivots between two clauses; waste. resolvent is tautology

- $(F \vee G)$ & $(\neg F \vee H)$ are antecedents.
- F → pivot.
- $(G \vee H)$ is resolvant.

* The aim of the proof system is to find the empty clause $\{\bot\}$. refutation proof system.

* if we want $\Sigma \vdash F$; (derive this!)

we use resolution proof system to bring $\Sigma \cup \{\neg F\} \vdash \bot$.

$a \rightarrow b \wedge b$

$a \cup \{\neg a \vee \neg b\} \rightarrow \ldots$

$\neg b$

Eg: $F = (p \vee q) \wedge (\neg p \vee q) \wedge (\neg q \vee r) \wedge (\neg r)$

$p \vee q \quad \neg p \vee q$

$q \quad \neg q \vee r$

$r \quad \neg r$

$\bot$ → F is UNSAT.

1) SAT
2) Proof — opposites
3) UNSAT

→ Implementation issues:-

We nneed to have some "blind rules" to cut down on unnecessary steps.

1) Superset clauses are redundant:

cause its DV() ∈ C. so waste...

if D ⊂ C ; then if ⊥ is derivable by C; then same with D.

2) Ignore valid/tautology clauses:-

if clause has P & ¬P; then its a tautology. Ignore it.

no loss of completeness.

3) Pure literal:-

¬p is a literal! Not an atom.

if a literal occurs in CNF; with its negation absent;

then keep the pure literal as 1 & ignore all its clauses.

(satisfiability preserved)

4) Unit clause propogation:-

if 'l' occurs as a clause ; then remove '¬l' from every clause.

not "in a clause"

& ignore all valid clauses.

(l → 1).

$$\frac{l \qquad \neg l \vee D}{D}$$

5) Prefer working on similar clauses:-

we like our resolvent to be short.

(since we need empty clause finally).

→ Completeness of my proof system:-

is my proof system complete?.

I mean, given a Σ is unsat; can I get $\perp$ by resolving Σ?

(set of clauses now..)

lets see;

* $Res^n(\Sigma)$ is the set of clauses that are derivable from resolution proofs upto depth n from Σ.

formally:-

$$Res^0(\Sigma) \equiv \Sigma.$$

$$Res^{n+1}(\Sigma) \equiv Res^n(\Sigma) \cup \{c \mid c \text{ is a resolvent from clauses } C_1, C_2 \in Res^n(\Sigma)\}$$

Eg: $\{(p \vee q), (\neg p \vee q)\} \equiv \Sigma$

$Res^1(\Sigma) = \{(p \vee q), (\neg p \vee q), (q)\}$.

$Res^2(\Sigma) \equiv Res^1(\Sigma)$.

* Since, vars (Σ) is finite (okayish...); we can derive only finitely many clauses via resolution...

as some point $Res^{n+1}(\Sigma) = Res^n(\Sigma)$.

∴ Now; $Res^*(\Sigma) \triangleq Res^n(\Sigma)$

(saturated resolution clauses from Σ.)

Theorem 10.1:-

if a finite set of clauses, Σ, is unsat; then $\perp \in Res^*(\Sigma)$.

Proof?

we induct on the no. of variables in Σ.

Be like whoaa....

base case:-

$||vars(\Sigma)|| = 1$.

then $\{p, \neg p\} \in \Sigma$.

$\therefore$ $\dfrac{p \quad \neg p}{\bot}$ $\checkmark\checkmark$ $\bot$ is in $Res^*(\Sigma)$.

Induction step:-

* let the n+1 elements of Σ be $P_1, P_2, \ldots P_n \& p$.

The theorem holds for all formulas (UNSAT wale) with n elements.

Σ_0 = All clauses of Σ, which contain p.
↳ confused with $res^1(\Sigma)$, in quiz.

Σ_1 ≡ All clauses of Σ, which contain $\neg p$.

$\Sigma = \Sigma_0 \wedge \Sigma_1 \wedge \Sigma_*$

Σ_* ≜ remaining clauses. (ie. without $p, \neg p$).

And...

$\Sigma_0' \triangleq \{C - \{p\} \mid C \in \Sigma_0\}$

$\Sigma_1' \triangleq \{C - \{\neg p\} \mid C \in \Sigma_1\}$

m $\quad$ m(p→1)

$\Sigma_0' \models \Sigma_0$

$\Sigma_1' \models \Sigma_1$

m $\quad$ m(p→0).

CLAIM:- if $(\Sigma_0' \wedge \Sigma_*) \vee (\Sigma_1' \wedge \Sigma_*)$ is SAT; then Σ is SAT.

there is no 'p'. only P_1 to P_n.

Provable

take advantage of $p \to 1$ or $p \to 0$.

Hence;

if Σ is UNSAT; $(\Sigma_0' \wedge \Sigma_*) \vee (\Sigma_1' \wedge \Sigma_*)$ is UNSAT

meaning; individually;

And, by Ind. hypo each can give a resolution proof for '⊥'

$\Sigma_0' \wedge \Sigma_*$ is UNSAT

$\Sigma_1' \wedge \Sigma_*$ is UNSAT

case-i: if clauses of Σ_* alone produce $\perp$.

then we are done. the same clauses in Σ too!

Terrific!

case-ii: clauses of Σ_* not sufficient for $\perp$?

then $\Sigma_1' \wedge \Sigma_*$ produce $\perp$. $\Sigma_0' \wedge \Sigma_*$ produced $\perp$.

$\Rightarrow$ $\Sigma_1 \wedge \Sigma_*$ produce "$\neg p$" $\Sigma_0 \wedge \Sigma_*$ produces p.

{Provable!}

$\therefore$ finally; Σ produces $\perp$.

$\therefore$ our resolution proof system is COMPLETE. What a proof!

---

* All the below are equivalent.

1. $\Sigma \vdash F$ — Σ proves/derives F
2. $\Sigma \models F$ — Σ implies F.

] valid.

3. $\phi \in Res^*(\Sigma')$; where Σ' is CNF of $\Sigma \cup \{\neg F\}$.] UNSAT.

opposites

Proof:-

$\Sigma \vdash F$ can be shown. $\phi \in Res^*(\Sigma')$

$\Sigma \vdash F$ proof rules are sound.

$\phi \in Res^*(\Sigma')$ $\Sigma \vdash F$ Last Theorem.

finite to Infinite:-

For some $\Sigma \subseteq \Sigma''$ (iff) $\Sigma'' \vdash F$

Σ'' is an infinite set.

proved by "compactness" on next page. [if infinite set implies F; then a finite subset also implies F.

Thm 10.3:-

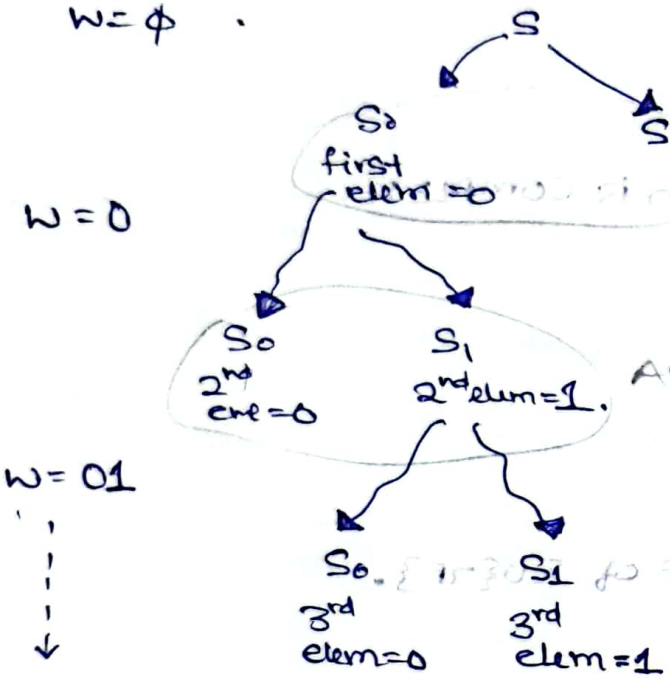
consider an infinite set S; of finite binary strings.

There exists an infinite string 'W' such that the following holds

$$\forall n. \; |\{ W' \mid W' \in S, W_n \text{ is prefix of } W' \}| = \infty$$

where W_n is prefix of W of length n.

Proof: By induction.



Compactness:-

A set Σ of formulas is sat iff every subset of Σ is sat.

maybe different model for each.

Proof:- forward ✓

backward.... use the above theorem.

(not easy

(if |Σ| is finite; then it is subset of itself

if |Σ| is infinite; use above theorem).

Proof in slides...

order variables; in accordance to formulas.

(order formulas in some arbitrary order).

* Thm 10.5:-

if Σ is a finite set of formulas; $\Sigma \models F$ is decidable.

(due to truth tables).
but exponential growth.

- effectively enumerable:-

if we can enumerate a set, using an algo; then its called effectively enumerable.

- Semi-decidable:-

A yes/no problem is semi-decidable; if we have an algorithm, only for one-side of the problem.

→ Thm 10.6:-

if Σ is effectively enumerable; $\Sigma \models F$ is semi-decidable.

Proof:-

let $G_1, G_2, \ldots$ be the enumerations (formulas) of Σ.

let $S_n = \{G_1, G_2, \ldots G_n\}$

we said; if $\Sigma \models F$; $\exists \Sigma_0 \subseteq \Sigma$; $\Sigma_0 \models F$.

$\exists K$; $\Sigma_0 \subseteq S_K$.

if $\Sigma_0 \models F$; $S_K \models F$.

enumerate S_n; & check $S_n \models F$ (which is decidable).

eventually; we'll say yes if $\Sigma \models F$ is true.

but; non-terminating; if $\Sigma \models F$ is false.

∵ semi-decidable.

Basically;

to say yes; we show existance of one model. fine

to say no; we gotta go over all ∞ models. not fine.

So; semi-decidable.

Lecture -11:- SAT SOLVERS :-

* we assume that input to a SAT Solver is in CNF.

(use tseitin's encoding to avoid blowup).

> introduce fresh vars

→ DPLL:- Davis-Putnam-Loveland-logemann method.

- partial model; we call elements of $Vars \to \mathbb{B}$ as partial models.
- State of a literal:

* under m; a literal l is true; if $m(l)=1$

P or $\neg p$ false; if $m(l)=0$

otherwise l is unassigned.

- State of a clause:

* under partial model m;

clause C is true; if $\exists l \in C$ s.t. $m(l)=1$.

is false; if $\forall l \in C,\ m(l)=0$.

otherwise C is unassigned.

- State of a formula:-

* Formula F is true, if $\forall C \in F;\ m(C)=1$.

is false, if $\exists C \in F;\ m(C)=0$.

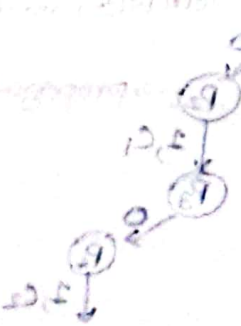
otherwise F is unassigned.

- unit clause:-

* in clause C; if $\exists l \in C$ such that l is unassigned & all others are 0;

then C is unit clause

l is its unit literal.



DPLL:

1. maintains a partial model (initially == ϕ)
2. Assigns unassigned variable 0 or 1 randomly.
3. Forced assignments due to unit literals.

Algorithm DPLL(F)

input: CNF F output: SAT/UNSAT

return DPLL(F, ϕ)

Algorithm DPLL(F, m)

input: CNF F, partial model m output: SAT/UNSAT

> if F is true under m, return SAT;

> if F is false under m, return UNSAT; // for backtracking purpose.

> if $\exists$ unit literal p under m then
  return DPLL(F, m[p→1]);

> if $\exists$ unit literal $\neg$p under m, then
  return DPLL(F, m[p→0]);

← unit propagation.

> choose an unassigned variable p & a random b ∈ {0,1}; // random decision.

if DPLL(F, m[p→b]) == SAT, then
  return SAT;

else
  return DPLL(F, m[p→$\neg$b]);

where is the "conflict node"?

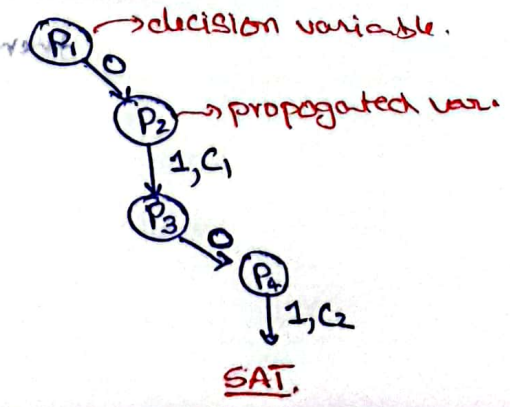
---

* Hence DPLL means:

- Decision
- Unit propogation] saves us the exp growth sometimes.
- backtracking

Eg: $C_1 = (P_1 \vee P_2)$

$C_2 = (P_3 \vee P_4)$

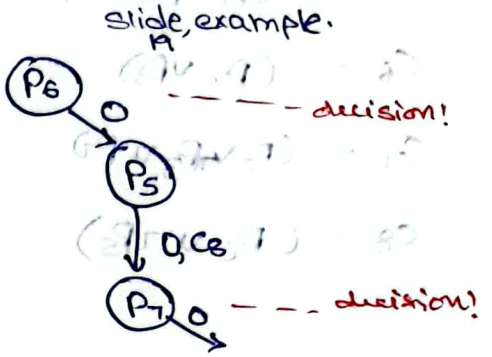


→ **Optimizations:-** (DPLL offers many optimizations)

- clause learning
- 2-watched literals
- ⋮

→ **Clause learning:-** over next page.

- current partial model; a run of DPLL.



- decision level of a true literal is the number of "random decisions" after which the literal was assigned true.

$\neg P_6$@1, $\neg P_5$@1, $\neg P_7$@2.

true literals

becoz, unit propogation isn't random decision.

is a run.

not a big deal. the usual DAG, which shows

→ **Implication graph during DPLL:-** (a,b) if $a \Rightarrow b$.

we maintain the following data structure:-

>

Under partial model m, the Implication graph is a labelled DAG (N,E) where, fine!

1. N is the set of true literals & a conflict node.

not atoms! By unit propogation; we get 0, 1 both. (via. two different paths)

2. $E = \{(l_1, l_2) \mid \neg l_1 \in \text{causeClause}(l_2)$ $l_2 \neq \neg l_1\}$

$(l_1 \Rightarrow l_2) \iff (\neg l_1 \vee l_2)$ is a clause.

$\text{causeClause}(l_2) \triangleq$ { clause due to which unit propogation made l_2 true. ϕ if l_2 is a decision variable

a function!

* Also annotate, each node with decision level.

Eg: let $(\neg a_1 \vee \neg a_2 \vee \neg \cdots \vee \neg a_n \vee b)$ be the causeClause(b). is a clause.

$\Rightarrow \neg a_1 = \neg a_2 = \cdots \neg a_n = 0;\ b = 1$

$(a_1 \wedge a_2 \wedge \cdots a_n) \Rightarrow b$ $a_1 = a_2 = \cdots a_n = 1$.

& we will have edges $(a_i, b)\ \forall i \in [n]$, in implication graph.

Eg:

$C_1 = (\neg P_1 \vee P_2)$

$C_2 = (\neg P_1 \vee P_3 \vee P_5)$

$C_3 = (\neg P_2 \vee P_4)$

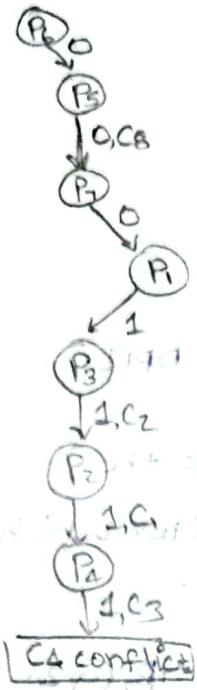
$C_4 = (\neg P_3 \vee \neg P_4)$

$C_5 = (P_1 \vee P_5 \vee \neg P_2)$

$C_6 = (P_2 \vee P_3)$

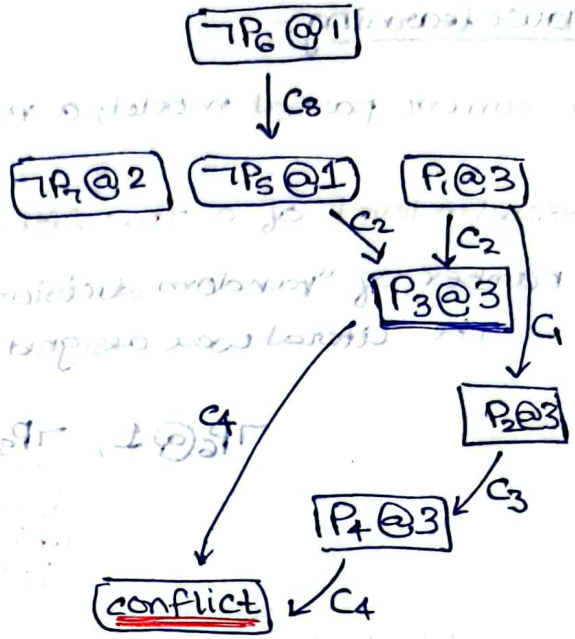
$C_7 = (P_3 \vee \neg P_3 \vee P_7)$

$C_8 = (P_6 \vee \neg P_5)$



Implication graph:-

- DAG. vertices are all true literals.
- we draw implication edges.



→ conflict clause:-

* we travel the implication DAG backwards, from conflict node, to find the "decisions" that led to conflict.

> The clause of negation of causing decisions is called conflict clause

∴ above $\neg P_6, P_1$ are decisions.

∴ $(P_6 \vee \neg P_1)$ is the conflict clause.

clause learning heuristics!

* Now; add conflict clause in the input clauses and backtrack to last Second decision & proceed DPLL.

conflicting

Theorem 4.1:-

Adding a conflict clause

1. doesnot change the set of satisfying assignments.
2. implies that the conflicting partial assignment won't be tried again.

* many clauses can satisfy above both properties.

→ Functional def" of conflict clause:- If a clause satisfies above both properties, we will say it is a conflict clause.

* conflict clauses:

1. Prunes away search spaces.
2. Records past work of the solver. (! So powerful).
3. Not complicated. Fits right into CNF.

→ CDCL algorithm. (Conflict Driven Clause Learning)

(DPLL + clause learning).

But the essence of this algorithm.

Algorithm : CDCL

Input: CNF F

decision level. decision stack. keeps count of how many variables were assigned at what dlevel (to back track).

$m := \emptyset$; $dl := 0$; $dstack := \lambda x.0$;

$m :=$ UnitPropogation(m, F); → assign all possible variables. not a single step...

do

  // backtracking

  while $m \not\models F$ do

    if $dl = 0$ then return UNSAT; !

    $(C, dl) :=$ AnalyzeConflict(m, F);

    m.resize(dstack(dl)); backtracking. $F := F \cup \{C\}$; clause learning

    $m :=$ UnitPropogation(m, F);

  // Boolean decision

  if F is unassigned under m then

    dstack(dl) = m.size();

    $dl = dl + 1$; $m :=$ Decide(m, F);

    $m :=$ UnitPropogation(m, F);

while F is unassigned under m or $m \not\models F$;

return SAT;

1) unitpropogation:-
applies unit propogation as many times as required.

2) Decide:-
gives a boolean value to an unassigned var

3) AnalyseConflict:-
returns a conflict clause and a 'dl' for backtracking.
if current dl = 3;
it will give back 2.

Lec 12:- SAT encoding:-

- Since SAT is np-hard; we can encode any NP-hard problem into SAT; in polynomial size.

* we can solve NP-hard problems using SAT solvers.

Objective of encoding:-

- compact encoding. (linear, if possible)
- redundant clauses may help the solver. (yeaa..) conflict clauses are redundant;
- encoding should be compatible with CDCL. conflict driven clause learning.

* CNF is our choice.

1. Most problems specify collection of restrictions on solutions. (conjunction)
2. And the restrictions are

if ( ) then ( ). (disjunction)

if ( ) then ( )
$\neg() \Rightarrow ()$ Sweet
$\neg() \vee ()$

Naturally; they are written into CNF form.

1. Graph Coloring:-

color a graph $(\{v_1, v_2, v_3, \ldots v_n\}, E)$ with atmost d-colours such that if $(v_i, v_j) \in E$ then the colours are different.

Encoding:-

v_{ij} is a boolean; $i \in [n]$; $j \in [d]$.

v_{ij} is 1; implies that v_i is having color j.

clauses:-

➤ each vertex has atleast one color

$$(v_{i1} \vee v_{i2} \vee v_{i3} \vee \ldots v_{id}) \quad \forall i \in [n]$$

But mind you! we didn't encode "atmost 1 color" here.

➤ if $(v_i, v_j) \in E$; then color should not be same.

i.e. $\forall k$; $(v_{ik} \Rightarrow \neg v_{jk})$

i.e. $(\neg v_{ik} \vee \neg v_{jk}) \quad \forall k \in [d]$.

2. Piegon Hole Principle:-

if we place n+1 pegions in n holes; there is a hole with 2 pigeons atleast.

clauses:-

P_{ij} be that i^{th} pegeon is in j^{th} hole.

> each pigeon should sit in "atleast" one hole.

$$(P_{i1} \vee P_{i2} \vee \cdots P_{in}) \quad \forall i \in [n].$$

> every hole should have atmost 1 pegion.

$$(P_{iK} \Rightarrow \neg P_{jK}) \quad \forall K;\ \forall i,j,\ i<j$$

$$\therefore (\neg P_{iK} \vee \neg P_{jK}) \quad \forall K;\ \forall i,j\ \ i<j.$$

12.2 Cardinality Constraints:-

$$\underbrace{P_1 + P_2 + \cdots P_n}_{\text{boolean variables \& algebric sum. "counting".}} \bowtie K. \quad \bowtie \in \{<, >, =, <=, >=\} \cup \{\neq\}$$

I forgot this before.

1) Encoding $P_1+P_2+\cdots P_n = 1$:- (could have done this in graph coloring)

> Atleast one of P_i's is true

$$(P_1 \vee P_2 \vee \cdots P_n)$$

> Not more than one is true!

$$(\neg P_i \vee \neg P_j) \quad \forall i,j \in i<j$$ But quadratic boom! ☹

→ sequential encoding:- we'll do better than quadratic.
by introducing fresh vars.

S_i be the variable, to indicate that count by i has reached 1.

* This encodes $P_1+P_2+\cdots P_n \leq 1$:-

$$(P_1 \Rightarrow S_1) \wedge \bigwedge_{1<i<n} \underbrace{((P_i \vee S_{i-1}) \Rightarrow S_i)}_{\text{for } S_i} \wedge \underbrace{(S_{i-1} \Rightarrow \neg P_i)}_{\text{for } P_i} \wedge (S_{n-1} \Rightarrow \neg P_n)$$

→ Bitwise encoding of $p_1+p_2+\cdots+p_n \leq 1$:-

$m = \lceil \log_2 n \rceil$

> consider bits $r_1, r_2 \cdots r_m$ → variables. log n more new vars.

> for each $i \in [n]$; let $b_1 b_2 \cdots b_m$ be the binary encoding of $i-1$.
not variables!

we want to add the constraint that p_i is 1.

$$(p_i \Rightarrow ((r_1 = b_1) \wedge (r_2 = b_2) \wedge \cdots (r_m = b_m)))$$

Eg:- say $r_1 = 1, r_2 = 0$ → in SAT model. & $n = 3$

~~or rather, $b_1 = 1, b_2 = 0$~~ $\therefore m = \lceil \log_2 3 \rceil = 2$

∴

now for SAT; p_1 ×

p_2 ×

p_3 ✓

$\therefore p_3 = 1$ & $p_1, p_2 = 0$.

What I wrote is correct!

$r_1 - r_m$ are new vars.

$b_1 - b_m$ are not vars.

our encoding will be

** $(p_1 \Rightarrow (r_1 = 0 \wedge r_2 = 0)) \wedge (p_2 \Rightarrow (r_1 = 0 \wedge r_2 = 1)) \wedge (p_3 \Rightarrow (r_1 = 1 \wedge r_2 = 0))$

in SAT model; $\{r_1 r_2\}$ determines which one of p_1, p_2, p_3 is 1.

Note:-
take it like this.

for each i; $b_{i1}, b_{i2}, \cdots b_{im}$ represent binary encoding of decimal "$i-1$".

so; if $i = 5$ & $m = 4$.

$5 \equiv 0101$; $i-1 \equiv 0100$

$\therefore b_{i1} = 0$ $b_{i2} = 1$, $b_{i3} = 0$; $b_{i4} = 0$.

→ encoding $P_1+P_2+\cdots P_n \leq K$:-] with this; I can encode $P_1+P_2+\cdots P_n \geq K$, also na.

- Generalised pairwise
- Sequential counter
- Operational encoding
- Sorting Networks
- Cardinality networks

] dumb.

1) Generalised pairwise:-

No K+1 variables must be true simultaneously;

write that! ^nC_K terms!

Foreach $i_1, i_2 \cdots i_{K+1}$

$(\neg i_1 \vee \neg i_2 \vee \cdots \vee \neg i_{K+1})$ like this; ^nC_K clauses

insane! not quadratic!

2) Sequential counter encoding:-

Let variable S_{ij}; $i \in [n]$; $j \in [K]$; denote the sum upto i terms has reached j or not;

clauses:-

remember! redundant clauses help the SAT solver!

> for first variable.

$$(P_1 \Rightarrow S_{11}) \wedge \bigwedge_{j \in [2,K]} (\neg S_{1j})$$

> for $P_i \& S_{ij}$:- $i > 1$

$$((P_i \vee S_{(i-1)1}) \Rightarrow S_{i1})$$

$$\wedge \bigwedge_{j \in [2,K]} ((P_i \wedge S_{i-1\,j-1}) \vee S_{(i-1)j} \Rightarrow S_{ij})$$

add+1

> more, helper constraints:-

$$(S_{(i-1)K} \Rightarrow \neg P_i)$$

the sum is reached!

since; no more ones.

3) operational encoding:-

Sum the bits using fulladders. compare the resulting bits against K. their expressions!.

- produces O(n) encoding.
- But not good for solvers;
  because its NOT - ARC - CONSISTENT.

think

this gives

$P_1 + P_2 + \cdots P_n = K$

I think.

→ ARC-CONSISTENCY:-

let $C(P_S)$ be a problem with variables $P_S = P_1, P_2, \cdots P_n$.

let $E(P_S, T_S)$ be encoding of problem; where $T_S = t_1, t_2, \cdots t_k$ are introduced.

Def^n 12.1:-

we say $E(P_S, T_S)$ is arc-consistent, if for any partial model m of E:-

1. if $m|_{P_S}$ is inconsistent with C; then unit propagation in E leads to conflict.
   projection

2. if $m|_{P_S}$ is extendable to m' by local reasoning in C, then unit propagation in E obtains m'' such that $m''|_{P_S} = m'$

i.e.

arc-consistency means

LOCAL REASONING ≡ UNIT PROPAGATION

! easy right!

i.e. unit propagation in E; must mimic Logical reasoning in C.

don't make life of solver hard.

Use SAT-solver for reason-calculus

Eg: $P_1 + P_2 + \cdots P_n \leq 1$ — Not Boolean calculator.

a)

An encoding is arc-consistent; if

1. At any step; if two pis are true; unit propagation should trigger conflict

(two pis are true: absurd by common sense. unit propagation should trigger conflict: encoding follows reasoning.)

2. If at any time;

one of P_i is true; unit propagation should make all others false.

(one of P_i is true: common sense says "All others are false". unit propagation should make all others false: unit prop. follows common sense.)

b) Consider $P_1 + P_2 + P_3 \leq 0$

use full adder-encoding:-] ... using SAT-solver as Bool calculator

our final clauses are:-

$$\neg(P_1 \oplus P_2 \oplus P_3) \wedge \neg((P_1 \wedge P_2) \vee (P_2 \wedge P_3) \vee (P_1 \wedge P_3))$$

(Since our RHS is 00. — SUM expression. — carry expression.)

But! My logical reasoning says, $P_1 = 0$, $P_2 = 0$, $P_3 = 0$ directly. 😵

But! without any "random decisions"; unit propagation is not able to give me a model.

unit propagation; not following logical reasoning.

123 :-

Sudoku:-

variables $V_{i,j,k} \in B$.

if $v_{ijk} = 1$,

$(i,j) = K$ in Sudoku.

> each cell is valid:-

$$\sum_{K=1}^{9} V_{i,j,K} = 1, \quad i,j \in \{1,2,\dots 9\}$$

> each row;

$$\sum_{i=1}^{9} V_{i,j,K} = 1 \quad j,K \in \{1 \dots 9\}$$

> each column;

$$\sum_{j=1}^{9} V_{i,j,K} = 1 \quad i,K \in \{1 2 \dots 9\}$$

> each grid:-

$$\sum_{s=1}^{3} \sum_{r=1}^{3} V_{3i+r,3j+s,K} = 1 \quad i,j \in \{0,1,2\} \; K \in \{1 2 \dots 9\}$$

Bounded model checking:-

I: vector of inputs

X: vetor of current state

O: vector of Outputs

X': vector of final state.

see example in video.

Prove: after 'n' steps; we get O, such that it satisfies some F(O).

let $T(I,X,O,X')$

then we do $T(I_0,X_0,O_1,X_1) \wedge T(I_1,X_1,O_2,X_2) \wedge \dots$

$\wedge \neg F(O_1,O_2,\dots O_n)$

if this is unsat; then no bugs.

First order Logic:-

quantifier over individuals.

not sets of individuals.

MSO-Logic.

* FOL syntax; three kinds of entities.
  - > variables (not boolean!)
  - > logical connectives
  - > non logical symbols: functions
    Predicates/Relations.

* Logical connectives now also include $\forall, \exists, =$

* Signature $S = (F, R)$
  - F: set of functions
  - R: set of predications/Relations.
  - each symbol has an 'arity!
    no. of inputs.

→ In propositional logic;

$$F = \emptyset \ \& \ R = \{P_1/0, P_2/0, \dots\}.$$

Propositional variables.

* $f/0 \in F$ is called constant.

1. S-terms Ts are given by

$$t ::= x \mid f(t, t, \dots t) \ ; \ x \in Vars. \ \& \ f/n \in F.$$

(n)

Variables VS constants:-

- variables: can be quantified
- constants: can't be quantified

2. S-atoms As are given by

$$a ::= P(t, t, \dots t) \mid t = t \mid \top \mid \bot$$

(n)

$t = t$ is an atom! ~~not~~ formula! too!
but also an atom!

3. S-formulas Ps are given by

$$F ::= a \mid \neg F \mid F \wedge F \mid F \vee F \mid \dots \mid F \oplus F \mid \forall F \mid \exists F$$

•precedance:-

$\forall \quad \neg \quad \exists$

$\vee \quad \wedge \quad \oplus$

$\rightarrow \quad \leftrightarrow$

(=) is for terms; not atoms. not predicates!

→ Model:-

> for signature $S=(F,R)$; a S-model m is a

$$(D_m; \{f_m : D_m^n \rightarrow D_m \mid f/n \in F\}; \{P_m \subseteq D_m^n \mid P/n \in R\}).$$

D_m: DOMAIN of m

each function map

each predicate map.

no assignment of vars here.

→ Assignment:-

An assignment is a map $Vars \rightarrow D_m$.

∴ now; we can define $m,v : T_S \rightarrow D_m$.

* F is satisfiable if there are m,v such that $m,v \models F$.

F is true in m (i.e. $m \models F$); if $\forall v$; $m,v \models F$.

F is valid if $\forall m,v$; $m,v \models F$.

* clubbing same quantifiers!

- Subterm.

no "Subatom".

- Sub formula.

1) closed terms / ground term.

A closed term is a term without any variables.

use constants! lot.

Eg: $F = \{f/1, c/0\}$

'c' is a closed term.

f(c) is one too!

H(c) is not a closed term. Bcoz, its NOT a term!.

Oh gawd!

2) Quantifier-free formula:-

formula, with no quantifiers.

Eg: H(c) ✓

f(x) ? Not a formula.

3) Free variables:-

Should occur somewhere in F; other than as quantifiers.

Eg: $\forall x\, F(x,y)$ is. y → free variable.
x → not free.

4). Sentence:-

A formula; with no free variables; is a sentence.

Literally, WOW!

NOTE: only a SENTENCE can be can be converted into FOLCNF.

→ FOL & counting:-

> Consider $S = (\{3\}, \{E/2\})$

$F = (\forall x \neg E(x,x)) \wedge \exists x \exists y . E(x,y)$.

No model with one element domain, can satisfy F.

=> if $m, v \models F$; then $|Dm| > 1$.

* if I want F; such that $m, v \models F$ then $|Dm| \geq 2$;

$\forall x \neg E(x,x) \wedge \exists x \exists y \exists z . (E(x,y) \wedge E(y,z) \wedge E(z,x))$

construct like this.

I think we can also construct F; such that $|Dm| < n$.
But dunno for sure!

Theorem 15.1:-

No FOL sentence can express that all satisfying models are finite.

* But, we can give a statement; that is true only in infinite models.!

its negation; is also satisfied by some infinite models.

So; Thm 15.1 holds!

→ Substitution :-

* A substitution σ is a map from Vars → Ts. We will write $t\sigma$ to denote $\sigma(t)$.

* Natural extension of σ, from variables to terms!.

$$c\sigma \triangleq c$$

$$f(t_1, t_2, \ldots t_n)\sigma \triangleq f(t_1\sigma, t_2\sigma, \ldots t_n\sigma)$$

* Further; extend this to atoms.

$$H(t_1, t_2, \ldots t_n)\sigma \triangleq H(t_1\sigma, t_2\sigma, \ldots t_n\sigma)$$

$$T\sigma \triangleq T$$

$$\bot\sigma \triangleq \bot$$

$$(t_1 = t_2)\sigma \triangleq t_1\sigma = t_2\sigma$$

unification!!

* projection:

$$\sigma_x = \sigma[x \mapsto x]$$

→ substitution in formulas:-

a substitution σ is suitable wrt G_i, x ,

if $\forall x.G_i$; $\forall y \in FV(G_i), y \neq x$; $x \notin y\sigma$.

got it! Aaarsh!

- composition:-

$$(x\sigma_1)\sigma_2 = x(\sigma_1\sigma_2)$$

also $\sigma_1\sigma_2\sigma_3 \equiv (\sigma_1\sigma_2)\sigma_3 \equiv \sigma_1(\sigma_2\sigma_3)$.

FOL – formal proofs:- (lecture 18):-

* we will introduce: *old rules will work. we need new rules for quantifiers & equality.

1. ∀ intro
2. ∃ intro
3. ∀ elim
4. { ∃ elim / ∃-instantiation.
   won't remove ∃ symbol; rather, introduces one!
5. REFLEX
6. EQSUB

1. ∃ intro :-

if a fact is true about a term;

$$\exists\text{-INTRO}\ \frac{\Sigma \vdash F(t)}{\Sigma \vdash \exists y\, F(y)}\quad y \notin FV(F(z));\ F(z)\{z \to t\},\ F(z)\{z \to y\} \text{ are defined.}$$

for some z. choose your F(z)!

Eg: 1. $\{H(x)\} \vdash H(x)$ ASSUMPTION

2. $\{H(x)\} \vdash \exists y\, H(y)$ ∃-INTRO to 1.

BAD derivations:-

Eg 16.2

1. $\{x=1, y=2\} \vdash x \neq y$ Premise

2. $\{x=1, y=2\} \vdash \exists y\ y \neq y$ wrong; since $y \in FV(F(z))$.

Eg 16.1:

Don't make another free variable as bounded.

1. $\Sigma \vdash F(f(x), y)$

2. $\Sigma \vdash \exists y\ F(y, y)$ wrong.

Eg: 16.3:-

1) $\{\exists y.\ c \neq y\} \vdash \exists y.\ c \neq y$ Assumption.

2) $\Sigma \vdash \exists y\,(\exists y\ y \neq y)$ wrong, since $F(z) = \exists y\, z \neq y$ & $\{z \to y\}$ is not valid.

Eg:-

1. $\Sigma \vdash \forall x\, f(x) = x$ — premise.

2. $\Sigma \vdash \exists y (\forall x\, y = x)$ wrong!

bcoz $f(z) = \forall x\, z = x$

$\{z \to f(x)\}$ is not defined.

common sense bro.
Dont tamper quantified variables.
They aren't "free"!
hmmm...

---

2. ∀ Intro:-

* 1. A fresh variable x.
2. prove F(x).
3. Hence, ∀x F(x).

> if something is true about a variable, that is not reffered anywhere; true always.

its kind of duh! ... but formally... we state this.

$$\forall\text{-INTRO}\ \frac{\Sigma \vdash F(x) \quad ; \ y \notin FV(F(z))}{\Sigma \vdash \forall y F(y)} \quad \xi\ \boxed{x \notin FV(\Sigma \cup \{F(z)\})}$$

no reference!

Eg:-

1. $\{H(x)\} \vdash H(x)$

2. $\{H(x)\} \vdash \forall y H(y)$ false.

cause $x \in FV(\Sigma)$

F(z) should not have 'x'.
⇒ all 'x' should be substituted by y.

3) ∀-elim:-

$$\forall\text{-ELIM}\ \frac{\Sigma \vdash \forall x F(x)}{\Sigma \vdash F(t)}$$ "no strings attached!"

4) ∃-elim or ∃ instantiation:-

$$\exists\text{-ELIM}\ \frac{\Sigma \vdash F(x) \Rightarrow G}{\Sigma \vdash \exists x_0 F(x_0) \Rightarrow G}\quad x \notin FV(\Sigma \cup \{G, F(z)\})\quad x_0 \notin FV(F(z))$$

x is fresh variable, not a part of G.

F(z) shouldn't have x. ⇒ all the 'x' should be replaced.

rule is kind of duh... but formally, we write...

Eg:- prove $\phi \vdash \exists x (A(x) \wedge B(x)) \Rightarrow \exists x A(x)$.

1. $\{A(x) \wedge B(x)\} \vdash A(x) \wedge B(x)$ Assumption.
2. $\{A(x) \wedge B(x)\} \vdash A(x)$ ∧-ELIM to 1.
3. $\{A(x) \wedge B(x)\} \vdash \exists x A(x)$ ∃-INTRO to 2
4. $\phi \vdash \{A(x) \wedge B(x)\} \Rightarrow \exists x A(x)$. ⇒-INTRO to 3.
   Any one 'x', would imply RHS
5. $\phi \vdash \exists \{A(x) \wedge B(x)\} \Rightarrow \exists x A(x)$ ∃-ELIM to 4.
   if such an 'x' exists, RHS is true.

"if any door would let us in, the we say; if there exists a door, then I goin"

5) REFLEX and EQSUB:-

$$\text{reflex}\ \Sigma \vdash t = t$$

$$\text{EQSUB}\ \frac{\Sigma \vdash F(s)\quad \Sigma \vdash s = t}{\Sigma \vdash F(t)}$$

$$\text{EQSYM}\ \frac{\Sigma \vdash s = t}{\Sigma \vdash t = s}$$ derived

Eg: prove $\phi \vdash \forall x,y\ (x \neq y \vee f(x) = f(y))$

direct rel.:

1. $\phi \vdash f(x) = f(x)$.
2. $\{x = y\} \vdash x = y$
3. $\{x = y\} \vdash f(x) = f(x)$
4. $\{x = y\} \vdash f(x) = f(y)$

- EQSYMM $\dfrac{\Sigma \vdash s = t}{\Sigma \vdash t = s}$.

∃ intro, ∀-elim, EQ-SYMM } don't check. incoming variable reference. anywhere. already existing one.

∀ intro, ∃-elim } check incoming variable reference.

↳ ∃-INSTANCIATION

NOREFERENCE in Σ, F(z), G.

Lecture-17:- FOLCNF:-

* we can convert any FOL sentence (whether true or false) into a FOL-CNF form.

by the following transformations:-

1) Rename apart :- rename variables for each quantifiers (So that they are not same).

2) Negation normal form:- pushing negation inside

3) Prenex form:- pulling all quantifiers to absolute front

4) Skolemization: removing existential quantifiers
equiSAT formula obtained. not equivalent.

5) CNF transformation: turn the quantifier-free part into CNF.

6) Syntactical removal of universal quantifiers:
a CNF with free variables.

Step-1: renaming apart

Thm 17.1:

if $x,y \notin FV(F(z))$ then $\forall x F(x), \forall y F(y)$ are provably equivalent.

* A formula F is renamed apart, if no quantifier in F, uses a variable that is used by another quantifier (or) occurs as a free variable in F.

By prev. Thm, we can freely rename variables.

Step-2: Neg. normal form.

Thm 17.2:- $\Sigma \vdash \neg\exists x \neg F(x)$, then we prove $\Sigma \vdash \forall x F(x)$.

1) $\Sigma \cup \neg F(x) \vdash \neg\exists x \neg F(x)$ monotonic
2) $\Sigma \cup \neg F(x) \vdash \neg F(x)$ Assumption.
3) $\Sigma \cup \neg F(x) \vdash \exists x \neg F(x)$ $\exists$ intro to 2.
4) $\Sigma \vdash \neg\neg F(x)$ By contradiction to 1,3
5) $\Sigma \vdash F(x)$ Rev double Neg to 4
6) $\Sigma \vdash \forall x F(x)$ $\forall$-INTRO to 5.

Step-3:- Converting into prenex form:-

"No occurences, no issues"

Thm 17.3:

let x be variable, such that $x \notin FV(F)$. Then F, $\forall x F$, $\exists x F$ are all provably equivalent.

Thm 17.4:

if no occurence, we pull quantifiers to top.

if $x \notin FV(G_i)$ then

- $\exists x F(x) \wedge \exists x G$ and $\exists x(F(x) \wedge G_i)$ are provably equivalent.
- $\forall x F(x) \vee \forall x G$ and $\forall x(F(x) \vee G_i)$ are provably equivalent.

(univ. quantifier over disjunction)

* A formula F is in prenex form; if all the quantifiers occur as prefix of F.

The quantifier free suffix is called matrix of F.

Proving F from $\exists x F$:-

1. $\Sigma \vdash \exists x F$ premise
2. $\Sigma \cup \{F\} \vdash F$ Assumption.
3. $\Sigma \vdash F \Rightarrow F$ $\Rightarrow$-INTRO to 2
4. $\Sigma \vdash \exists x F \Rightarrow F$ $\exists$-ELIM to 3.
5. $\Sigma \vdash F$ $\Rightarrow$-ELIM to 1, 4.

Also note that;
existential quantifier is distributable over disjunction.

$\exists x(F(x) \vee G(x)) \equiv \exists F(x) \vee \exists x G(x)$

universal quantifier is distributable over conjunction.

$\forall x(F(x) \wedge G(x)) \equiv \forall x F(x) \wedge \forall x G(x)$

Q. Show equivalences:-

- $\forall x F \Rightarrow G \equiv \exists x(F \Rightarrow G)$

$\neg(\forall x F) \vee G$.

$\exists x \neg F \vee G$.

$\exists x(\neg F \vee G)$ G is

$\exists x(F \Rightarrow G)$

Step4- Skolemization :-

* Skolemization removes the $\exists$-quantifier, and only $\forall$-quantifiers are left.

- The replacement of $\exists$ by a function f is called Skolemization.

  And f is called Skolem function.

Thm 17.5:-

let F be a S-formula, $FV(F) = \{x, y_1, y_2 \cdots y_n\}$ and $f^n \in F$ doesn't occur in F. For each model m', there is another model m;

$$m \models \exists x.F \rightarrow F(f(y_1, y_2, \cdots y_n))$$

(for any assignment).

i.e. m, m' differ only in interpretation of f.

Proof:-

consider model/structure m'. we will construct m.

Now; let us construct an interpretation $f': D_{m'}^n \rightarrow D_{m'}$ of f as:

$$f'(d_1, d_2, \cdots d_n) \triangleq \begin{cases} d, \text{ if } m', \{y_1 \rightarrow d_1, y_2 \rightarrow d_2 \cdots y_n \rightarrow d_n\} \models \exists x.F \\ \quad \bullet \text{ choose } d \in D_{m'} \text{ such that} \\ \quad m', \{x \rightarrow d, y_1 \rightarrow d_1, \cdots y_n \rightarrow d_n\} \models F. \\ d, \text{ choose any } d \in D_{m'} \text{ if otherwise.} \end{cases}$$

What is the cost here?

lets define $m \triangleq m'[f \mapsto f']$.

as f doesn't occur in F, if $m, v \models \exists x.F$ then $m', v \models \exists x.F$

- Due to construction of m,

$$m, v \models F\{x \mapsto f(y_1, y_2, \cdots y_n)\}$$

Substitution.

Thm 17.6:-

let $F(x)$ be a (F,R)-formula with $FV(F) = \{x, y_1, y_2, \ldots y_n\}$ and $f/n \notin F$, f does not occur in $F(x)$.

$\forall y_1, y_2, \ldots y_n . \exists x . F(x)$ is Sat $\iff$ $\forall y_1, y_2, \ldots y_n \, F(f(y_1, y_2, \ldots y_n))$ is SAT.

How many universal quantifiers come before $\exists$, that many is the arity of the function.

Apply from outside to inside

Eg: $\exists x \, \forall y \, \exists w \, \forall z \, F(x,y,z,w)$

{ no $\forall$ before $\exists x$. Hence use $f_1/0$ i.e., $c/0$

$\forall y \, \exists w \, \forall z \, F(c,y,z,w)$

{ one $\forall y$ before $\exists w$. use $f/1$.

$\forall y \, \forall z \, F(c, y, f(y))$

Step 5,6:- convert the body into CNF. Drop explicit quantifiers.

use propositional logic to convert the quantifier-free suffix into CNF.

Note:-

we may use Tseitin encoding to obtain CNF, which introduces fresh propositional predicates of arity 0.

Is there a quantifier over those propositional predicates?

☹, quantifier, only over variables! Not predicates, Not constant functions!

Also... nothing like propositional "variables" in FOL.

* consider the skolemized prenix clauses

$$\forall x_1 x_2 \cdots x_n \; C_1 \wedge C_2 \wedge \cdots C_k.$$

Since $\forall$ distributes over $\wedge$; we have

$$(\forall x_1 \cdots x_n C_1) \wedge (\forall x_1 \cdots x_n C_2) \wedge \cdots (\forall x_1 \cdots x_n C_k)$$

We'll view this as conjunction of clauses

$$C_1 \wedge C_2 \wedge \cdots C_k.$$

Since we are dealing with sentences;

We'll understand that all free variables are universally quantified.

Lec·18:- Terms and unification

* The process of equating terms, by substitutions on variables; is called unification.

very chill concept.

Defⁿ 18.1:-

For terms t, u a substitution σ is a unifier of t, u if $t\sigma = u\sigma$.

Ex thus we say t, u are unifiable.

* $g(x_i)$, $f(x_i)$ are NOT UNIFIABLE.

its chill! don't again think what is g's map, f's map.

we don't work on a specific model here.

we work on a signature.

→ More general substitution:-

σ_1, σ_2 are substitutions; σ_1 is more general than σ_2 if there is a substitution τ, such that $\sigma_2 = \sigma_1 \tau$

we write $\sigma_1 \geq \sigma_2$

'in generality'

→ Most general unifier:- mgu not unique.

σ, a unifier of t, u is mgu, if it is more general than any other unifier.

Eg:

$f(x, g(y))$ and $f(g(z), u)$. are two terms.

1. $\sigma_1 = \{x \mapsto g(z), u \mapsto g(y), z \mapsto z, y \mapsto y\}$
2. $\sigma_2 = \{x \mapsto g(c), u \mapsto g(d), z \mapsto c, y \mapsto d\}$

c, d are constants.

Here $\sigma_1 \geq \sigma_2$.

↳ mgu. (we'll see, how to decide).

more general unifier → more scope in further analysis. more freedom.

Def 18.4:-

A substitution is renaming if $\sigma : vars \to vars$ & is one-one.

Thm 18.1:-

if σ_1, σ_2 are mgus of t, u, then there is a renaming τ such that $\sigma_1 = \sigma_2 \tau$.

mgu is unique, upto renaming.

Finding unifiers:-

→ We'll try to fix disagreement b/w two terms by substitution.

when no more disagreements → unified!

not able to fix → un-unifiable.

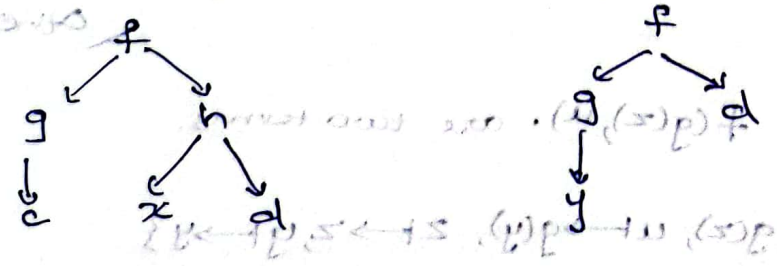
Defn 18.5:- Disagreement pairs

For terms t, u; d_1, d_2 are disagreement pair if

1. d_1, d_2 are subterms of t, u respectively.
2. the paths to d_1 in t, d_2 in u is same
3. roots of d_1, d_2 are different (disagree!)

Eg:

$t = f(g(c), h(x,d))$ $\quad u = f(g(y), d)$



(c, y) $(h(x,d), d)$ are disagreement pairs.

Robinson's Algorithm for computing MGUs:- pretty naive:-

Algorithm 18.1: MGU($t, u \in T_\Sigma$)

$\sigma := \{\}$,

while $t\sigma \neq u\sigma$ do

  choose disagreement pair d_1, d_2 in $t\sigma, u\sigma$;

  if both d_1, d_2 are non-variable terms, return FAIL;

  if $d_1 \in Vars$ then

    $x := d_1$; $s := d_2$;

  else

    $x := d_2$, $s := d_1$;

  if $x \in FV(s)$, then return FAIL;

  $\sigma := \sigma\{x \mapsto s\}$

return σ;

→ What about correctness of Robinson's Algo.-

* MGU(t, u) returns a unifier iff t, u are unifiable.
  Furthermore, σ is a mostgeneral unifier.

OKAY!
proved in slides.

~~Krishna mam~~:- Not Yet

Lec 19- FOL resolution.

* we apply resolution, when an atom and its negation are in two clauses.

$$\text{RESOLUTION} \quad \frac{FVC \quad \neg FVD}{CVD}$$

→ complication in FOL:-

the quantified variables might have different names.
(Remember! all are quantified? Sentence!)

we'll make two terms equal by unification.

↓

This'll reduce our scope of proof; but we'll be able to move ahead!

make the cases covered under proof smaller.

→ Three issues with unification:-

* Before proof rules, lets tackle the following

minor 1) Did we learn about unifying atoms?
nah! just terms. But... don't be a dk. Its okay!

major 2) Is substitution of variables, a valid operation under derivatn?

major 3) Handling of variables across clauses.

1) unifying atoms:-

- not a biggie! just see atom symbol as function symbol.
  predicate

2) deriving after substitution:-

- we know that the following proof is valid:

$\Sigma \vdash \forall x,y.\ F(x,y)$ — premise

$\Sigma \vdash F(t_1(x,y), t_2(x,y))$ — $\forall$-ELIM

$\Sigma \vdash \forall x,y.\ F(t_1(x,y), t_2(x,y))$ — $\forall$-INTRO.

* Hence we have, clauses;

$$\frac{C}{C\sigma}\sigma$$

this derivation is sound.

3) variables across clauses even though same name, are not same:-

recall! universal quantifiers distribute over conjunction:-

both are same

$\forall x(F(x) \wedge G(x))$

$\equiv$

$\forall x F(x) \wedge \forall x G(x)$

both x are different now!

there are hidden universal quantifiers; for every FOLCNF expression Finally; its a sentence.

∴ even though written same name, we view the variables occuring in different clauses as different.

within a clause → treat as SAME.

this'll help us; if we have like

$\neg R(x) \vee R(f(x))$

we think we can't unify them

but we can! $\neg R(x_1) \vee R(f(x_2))$ ✓ unify!.

Our satSolver(FOL) output should be a model → def^n for function, predicate, domain.

not a assignment.

no meaning of assignment, since no free variables

→ Resolution Theorem Proving:- (we've seen formal theorem proving)

Input: a set of FOL clauses F.

$$\text{ASSUMPTION} \quad \frac{}{C} \quad C \in F$$

$$\text{RESOLUTION} \quad \frac{\neg A \vee C \quad B \vee D}{(C \vee D)\sigma} \quad \sigma = mgu(A,B)$$

yea baby!

Eg: consider $P(x,y) \vee Q(y)$ and $\neg P(x,x) \vee R(f(x))$ as 2 clauses.

we, rewrite them as

$$P(x_1, y_1) \vee Q(y_1) \qquad \neg P(x_2, x_2) \vee R(f(x_2))$$

now; unifier: $(x_1 \mapsto x_2, y_1 \mapsto x_2)$

∴ finalclause is

$$Q(x_2) \vee R(f(x_2)) \checkmark$$

* Why MGU? why not any unifier?

once a clause becomes specific, we cannot go back.

why not keep it general.

we'll have maximum opportunity to find the ⊥ clause.

→ Rule: FACTORING:-

A clause may itself have copies of facts, that can be unified:-

$$\text{FACTOR} \quad \frac{L_1 \vee L_2 \vee \dots L_k \vee C}{(L_1 \vee C)\sigma} \quad \sigma = mgu(L_1, L_2 \dots L_k)$$

* This rule may appear superfluous, but is essential for FOL completeness

It captures the idea that same concept can be expressed in many ways.

Eg: Suppose we have $P(x) \vee P(y)$. This clause is not economical.

∴ we derive $P(x)$ as

$$\text{FACTOR}\ \frac{P(x) \vee P(y)}{P(y)}\ \sigma = \{x \mapsto y\}$$

Rule:- applying equality over clauses:-

$$\text{PARAMODULATION}\ \frac{s = t \vee C \qquad D(u)}{(C \vee D(t))\sigma}\ \sigma = mgu(s,u)$$

Rule:- Finishing disequality

$$\text{REFLEXIVITY}\ \frac{t \neq u \vee C}{C\sigma}\ \sigma = mgu(t,u)$$

Krishna Mam:-

First-order Logic:-

- A formalism to specify properties of mathematical structures like
  - graphs
  - partial orders
  - words
  - groups
  - rings
  - the world at large. Eg: every father is older than his child.

* Vocabulary or Signature τ, is a set, consisting of
  - > constants $c_1 c_2 .. c_n$
  - > Relations R_1, R_2 .. each with some arity denoted as R_i^k - arity

Eg: $\tau = (E^2, F^3)$

$t_1 = t_2$; equality is available, irrespective of Signature.

- Operator precedance: $\neg > \wedge > \vee > \rightarrow > \forall$

* Structure/model A of signature τ consists of
  1. A non empty set A or U(A) called universe
  2. for each constant c, a fixed element c_A is assigned from U(A)
  3. For each k-ary rel^n, R_i^k; a set of k-tuples from A^k is assigned.

- Structure A is finite, if U(A) is finite.

To make sense of a formula, we need structures.

Otherwise, they are just symbols. without meaning.

1. A total order structure:-

$\tau = \{<, S\}$ with $<, S$ as binary.

- a finite order structure $\mathcal{O}$ is $\{O, <^{\mathcal{O}}, S^{\mathcal{O}}\}$ → the total order set
- $<^{\mathcal{O}}$ is the ordering on O.
- $S^{\mathcal{O}}$ is the ~~ord~~ successor on O.

2. Word Structure: (Word structure! not Word signature!).

Word → sequence of symbols over a (finite alphabet)

Alphabet $\Sigma = \{a, b, c\}$

Eg: $\underset{\text{Signature}}{\tau} = \{<, S, Q_a, Q_b\}$ <, S are binary; Q_a, Q_b are unary

Word structure $\underset{W}{=} \{\underset{\text{universe}}{U(W)}, <^W, S^W, Q_a^W, Q_b^W\}$

- $U(W)$ consists of positions in a word W, over symbols a, b.
  0 to len-1
- Q_a^W → set of positions, labelled as 'a'.
- Q_b^W — — — — — — — — as 'b'.

$<^W, S^W$ is total ordering on [0, len-1].

→ each structure defines a unique word:-

$U(W) = \{0 1 2 \cdots 8\}$

$Q_a^W = \{0 1 4 6 8\}$ $Q_b^W = \{2 3 5 7\}$

$<^W = \{(0,1)(0,2)\cdots (7,8)\}$ $S^W = \{(0,1)(1,2)\cdots(7,8)\}$.

ordering on positions ☺.

defines aababababa.

* $\forall x \left(Q_b(x) \rightarrow \exists y (x < y \wedge Q_a(y)) \right)$

think in english...

for every 'b', there is an 'a' after it.

↓

defines a 'language'.

→ Satisfiability:-

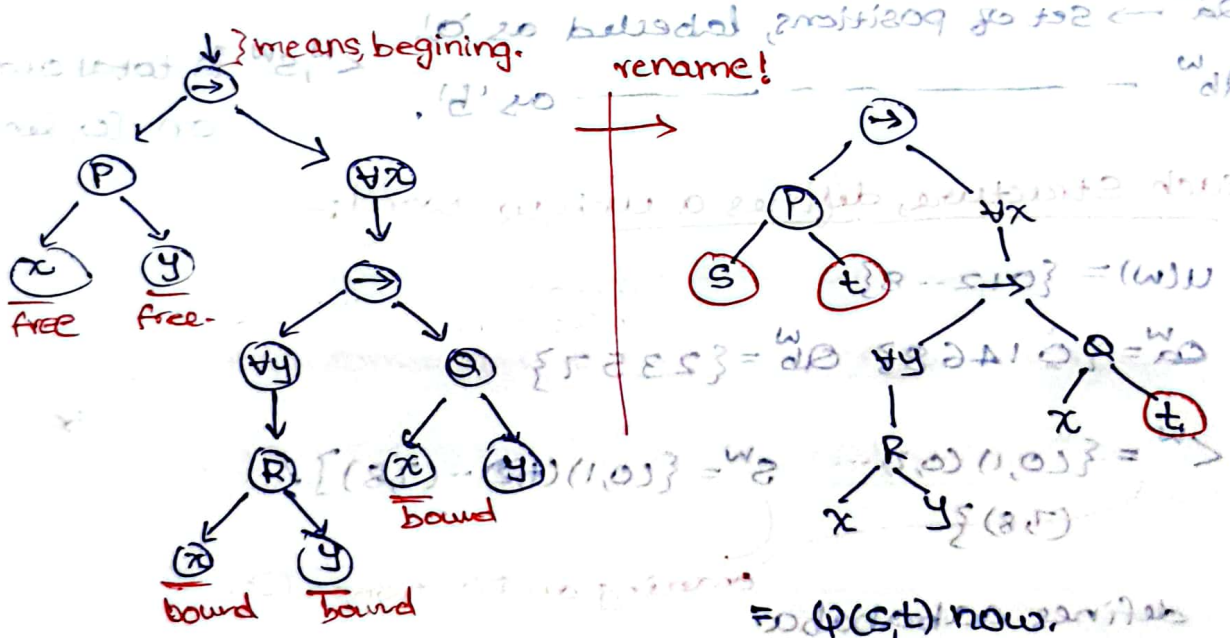
* for a PL formula → sat means existence of an assignment of bools.

satisfiability for FO formula ψ over signature τ;

means existance of a structure A of τ, such that ψ is true on A.

- Given ψ; denote it by $\psi(x_1, x_2, x_3, \ldots x_n)$ where these all $\in Free(\psi)$
- Some ψ may have x as a free variable and also bounded at someplaces.

$P(x,y) \longrightarrow \forall x(\forall y R(x,y) \rightarrow Q(x,y))$



$\Rightarrow \psi(s,t)$ now.

s,t are never bounded in the graph.

(formula)

* Assignments (α) on τ-structure.

assign every variable from $\mathcal{V}$, a value from $U(A)$.

- assignments matter, only to free variables.

- $\mathcal{A} \models_\alpha \varphi$; means φ is true over a structure $\mathcal{A}$, under assignment α.

* A formula φ is said to be satisfiable over signature τ; if for SOME structure $\mathcal{A}$, and SOME assignment α $\mathcal{A} \models_\alpha \varphi$.

  * said to be valid, iff for EVERY $\mathcal{A}, \alpha$ $\mathcal{A} \models_\alpha \varphi$. over sign. τ

  * formulae $\varphi(x_1, x_2, \ldots x_n)$ $\psi(x_1, x_2 \ldots x_n)$ are equivalent, if
    (mind you! Same variables)
    $\forall$ Structures, Assignments $\mathcal{A} \models_\alpha \varphi \iff \mathcal{A} \models_\alpha \psi$.

Eg: $\varphi_1(x) \triangleq \forall y R(x,y)$ $\varphi_2 \triangleq \exists x \forall y R(x,y)$

) are equisatisfiable.

Not a sentence.

careful → we're dealing with FO expressions, which aren't sentences.

∴ we say;

there is some assignment ! } would mean $\exists x$.
$\mathcal{A} \models_\alpha \varphi_1(x)$. ∴ would sat φ_2.

(lol)

Q) TRAP T/F: For a sentence φ, any any two assignments α_1, α_2

$\mathcal{A} \models_{\alpha_1} \varphi$ iff $\mathcal{A} \models_{\alpha_2} \varphi$.

True! "Sentence"! Assignments mean shit!

l_2.pdf

—× × ×—

L3.pdf:-

→ Satisfaction, Validity:-

* Given an FO formula $\varphi(x_1, x_2 \cdots x_n)$; over a signature τ;

  > Satisfiable; if there exist $\mathcal{A}, \alpha$ such that $\mathcal{A} \models_\alpha \varphi(x_1, x_2 \cdots x_n)$

  > valid, if for all $\mathcal{A}, \alpha$; $\mathcal{A} \models_\alpha \varphi(x_1 \cdots x_n)$

* But; in propositional logic; n bools → 2^n assignments → can decide satisfiability
  decidable.

  in FO logic; universe → arbitrary size → how to decide?

  ↳ SAT is undecidable. (proof, beyond course anta)
  i.e. if SAT; → then we'll somehow get
  if UNSAT; → our Algo won't terminate.
  Semi-decidable

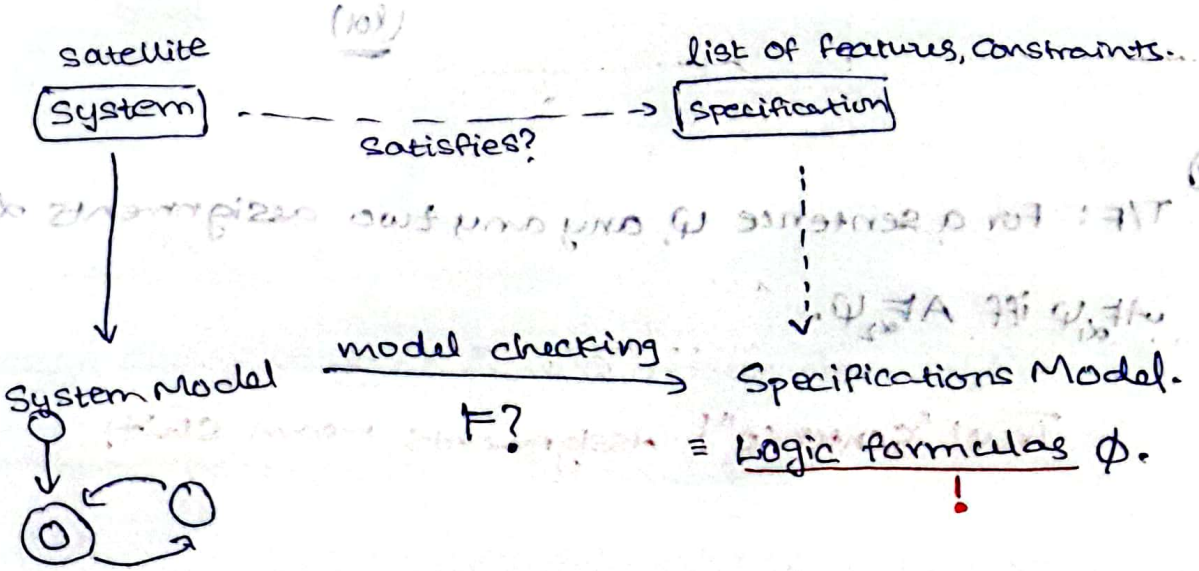
  To make FO-SAT decidable; lets fix the type of model/structure.
  (year further, it will come.)
  WORD STRUCTURE!

  But what's special about word structure?
  Any practical implication?

  *MODEL CHECKING!

Hence, FO-over WORDS. is what we learn.

*



→ Model checking:-

1. Abstract the given system (code/circuit) as a finite state transition system, G — not part of our course.

2. Behaviours of the system: — our course

sequence of actions, taken by G.

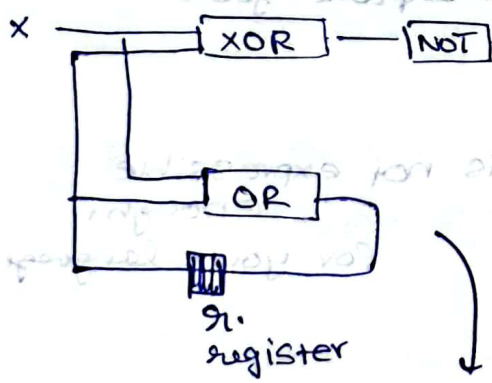
→ these will be our words. actions ≙ Alphabet.

3. Write the property of interest, as a Logic formula φ. — not our course

4. See. whether $G \models \varphi$. — our course

the regular difinition... whichever model/structure satisfies G, also satisfies φ.

Eg:



State ≙ {x, y, r}

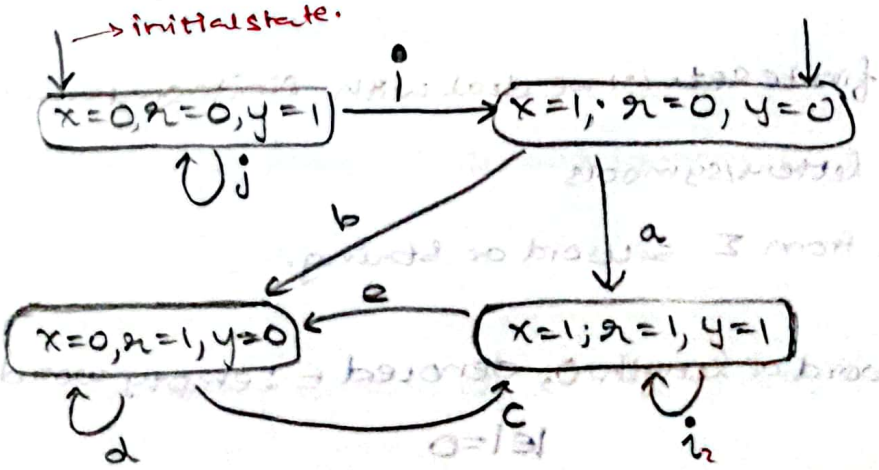
transition ≙ alphabets for our word structure.

{ i, j, a, b, c, d, e, i_2 }

FST Graph.

this is simple.

assume $r_{init.} = 0$



words!

possible behaviours:

jjiae.

jibddci_2

property:

1. No two actions can be i.

$\neg \exists x,y \, (x \neq y \wedge Q_i(x) \wedge Q_i(y))$

2. Every i, followed by a, or, b.

3. Give some φ.

check whether $G \models \varphi$.

→ FO over words:- Signature τ is fixed. $\{<, S, Q_a, Q_b, \dots\}$

* Given an FO sentence φ; over words; is it satisfiable?

  * There could be infinitely many words that satisfy φ.

  * $L(\varphi) = \{$ word structure $W \mid W \models \varphi\}$ is called language of φ.

    every structure, corresponds to a unique word. aah!

→ Expressiveness and satisfiability:-

* Signature for FO over words is fixed.

Expressiveness:-

- Given Language L; can you write a FO over words φ; $L(\varphi) = L$

  set of words.
  finite/infinite

- If we can, FO is "expressive enough" to capture your language/specification/property.

- if we can't have such an φ; then FO is not expressible enough, for your language

Satisfiability:

Given a FO: φ; is $L(\varphi)$ non-empty?

---

* An alphabet Σ is a finite set. (*: we deal with finite state machines)

  elements ≡ letters/symbols

  - sequence of symbols from Σ ≡ word or string.

* There is a unique word of length 0, denoted ϵ : empty word.

  $|\epsilon| = 0$

  $|w|$ = length of word.

- $aaaaa \equiv a^5$

  $a^0 \equiv \epsilon$

- Set of all words over Σ; denoted by Σ^*.

  ↳ infinite, but countable.

  $\{a\}^* = \{\epsilon, a, aa, \cdots\} = \{a^n \mid n \geq 0\}$

  $\{a,b\}^* = \{\epsilon, a, b, aa, ab, ba, ab, aaa, \cdots\}$

  ↳ except when $\Sigma = \{\}$.

By convention;

$\{\}^* = \epsilon$ quizquestion!

i.e. $\phi^* = \epsilon$

$\{a\}^* = \{\epsilon, a, aa, \cdots\}$

$\{a\}^+ = \{a, a^2, a^3, \cdots\}$

- $\{a,b\} = \{b,a\}$. but $ab \neq ba$.

- ϕ is the set, with no words.

  $\phi^* \equiv \{\epsilon\}$ is the set, with 1 word. } ★ crazy.

- ϵ is the identity for concatenation

  $x \cdot \epsilon = \epsilon \cdot x = x$.

* x^n: concatenating x - n times.

$(aab)^2 = aabaab$

* $|x|_a$ = no. of times a occurs in word x.

$|\epsilon|_a = 0$

* improper prefixes of $aab = \{\epsilon, aab\}$

* Given finite alphabet Σ;

- Subsets of Σ^* are called languages.

let $A, B, C \subseteq \Sigma^*$.

- $A \cup B = \{x \in \Sigma^* \mid x \in A \mid x \in B\}$
- $A \cap B =$ -- ✓✓
- $\bar{A} = \{x \in \Sigma^* \mid x \notin A\}$
- $AB = \{xy \mid x \in A, y \in B\}$

cartesian product.

concatenation

* For a language $A \subseteq \Sigma^*$;

- $A^0 = \{\epsilon\}$
- $A^{n+1} = A \cdot A^n$

cartesian product.

- $A^* = A^0 \cup A^1 \cup A^2 \cup \cdots$

Languages; not words.

- $A^+ = AA^* = A \cup A^2 \cup A^3 \cup \cdots$

* $A \cup \phi = \phi \cup A = A$.

$A \cup \{\epsilon\} \neq A$ need not be!

* → $A \cdot \{\epsilon\} = \{\epsilon\} \cdot A = A$

$A \cdot \phi = \phi \cdot A = \phi$

— IMP. DIFFERENCE.

Results:-

for Languages (Sets)

1) union, intersection distribute over union:-

? mam wrote.

$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$$

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$$

2) Concatenation distributes over union:-

dont confuse with union....

$$A(B_1 \cup B_2) = AB_1 \cup AB_2$$

3) concatenation doesn't distribute over intersection:-

$$A(B \cap C) \neq AB \cap AC.$$

{a}

{ε, a} {ε, b} understandable.

---

* L = words that have ~~no two consecutive a's~~ even length.

FO: φ = ? such that $L(\varphi) = L$.

* FO is NOT enough to capture language L.

we need 2nd order logic.

FO → quantifiers over individual element.

SO → FO + { something (set of elements.) else }.

---

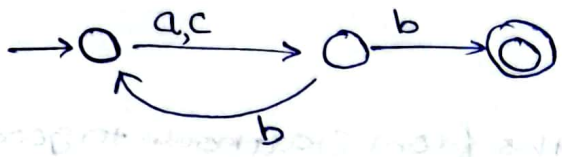
* we'll convert FO: formula to something familiar; to deal with satisfiability.

automatons?

→ Automatons:-

see how they fix the alphabet.

* Given FO formula φ over an alphabet Σ, construct an edge labelled graph G_φ: a graph whose edges are labelled by Σ.



* every path in G_φ → word on Σ.

* G_φ has special kinds of vertices:-
  - Unique vertex called start vertex. (has an '↓' over it).
  - Some vertices called good vertices. (double circle).

* read words from paths from start vertex to any good vertex.

  Call this set $L(G_\varphi)$.

  language of graph.

... Ensure that G_φ is constructed, so that $L(\varphi) = L(G_\varphi)$.

But, can we? Yes!

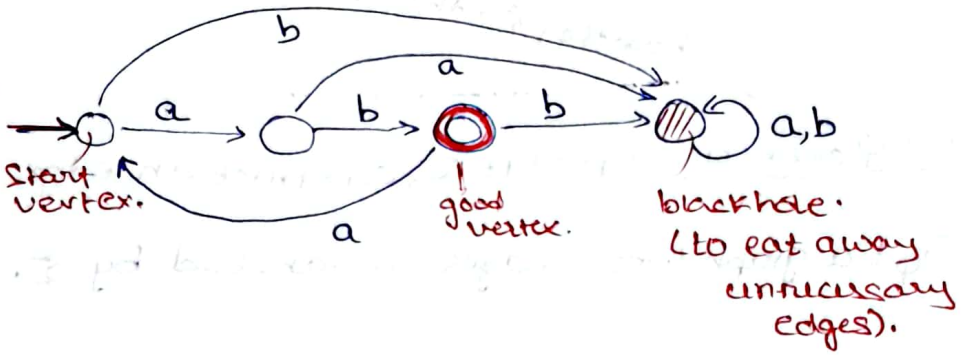
then

$sat(\varphi) \triangleq$ reachability of some good node from start node.

do DFS....

Hence; we decide on SAT of FO over WORDS.

*



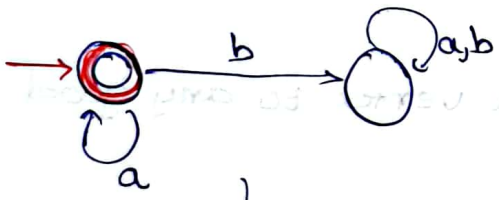
- vertices of graph → States.
- graph accepts words along paths from start ~~node~~ state to good state.
- Set of accepted words ≡ language (G).

Now; language is $ab, abaab, \cdots$

$= ab(aab)^*$

- Can we give ψ; such that $L(\psi) = L(G)$?

*



language is $\{\epsilon, a, a^2, \cdots\}$

a^*.

- & ψ: $\forall x\, Q_a(x)$; then $L(\psi) = a^*$ +∞!

* These "graphs" are called Deterministic, Finite State Automaton.

DFA.

$A = (Q, \Sigma, \delta, q_0, F)$

Q: Finite set of states

Σ: finite alphabet.

δ: $Q \times \Sigma \to Q$ is the transition function. → hence "Deterministic"! we know where to go!

q_0: $\in Q$, Start state

F: $\subseteq Q$, set of good states

Undeterministic!

$Q \times \Sigma \to$ subsets of Q.

2^Q.

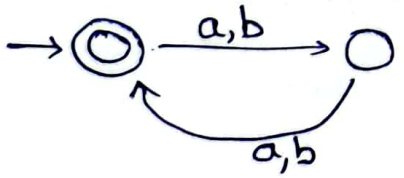
$L(A)$: all words leading from q_0 to some $f \in F$.

* A language $L \subseteq \Sigma^*$ is called regular iff there exists some DFA: A; $L = L(A)$

A language $L \subseteq \Sigma^*$ is called FO-definable, iff there exists some FO over words (φ); $L = L(\varphi)$

→ Is it regular? is it FO-definable?

1. $\Sigma = \{a, b\}$, L = even length words.



∴ L is regular.

But not FO-definable :C.

we need 2nd order logic.

$L = \{a^n b^n \mid n \geq 1\}$

is not FO definable

not regular.

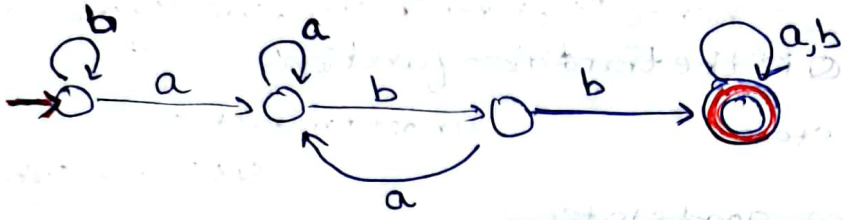
2) $\Sigma = \{a, b\}$

L: all words that contain abb

2.1) $\psi = \exists x,y,z\,(Qa(x) \wedge Qb(y) \wedge Qb(z) \wedge S(x,y) \wedge S(y,z))$ wow!

L is FO-definable

2.2)



L is regular.

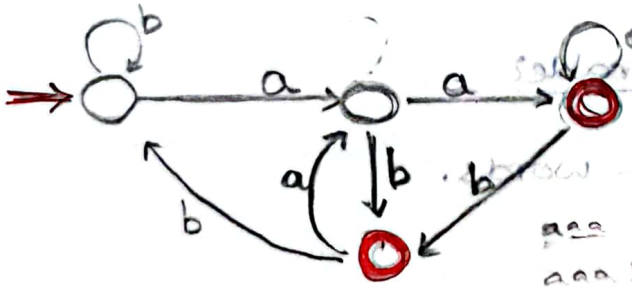
(will G be unique?)

3) $\Sigma = \{a, b\}$

L: words, with last second letter as 'a'.

L is FO-definable ✓✓.

lets attempt graph....



would have been easier in NFA

L is regular.

* we plan to show that

L is FO definable $\Rightarrow$ L is regular.

converse doesnot hold! (L is words with even length).

→ extended state transition function ($\hat{\delta}$):-

$\hat{\delta} : Q \times \Sigma^* \to Q$; extension of δ to strings, rather than just letters.

$\hat{\delta}(q, \epsilon) = q$

$\hat{\delta}(q, wa) = \delta(\hat{\delta}(q, w), a)$

→ Bucket interpretation for States:-

for any $w \in \Sigma^*$; $\hat{\delta}(q_0, w) \in Q$. & since this is DFA;

$\hat{\delta}(q_0, w)$ is unique value.

→ every word is mapped to some state.

→ every state 'contains' a subset of Σ^*.

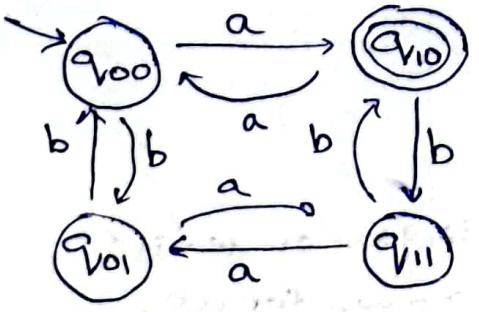
States partition the set Σ^*.

∴ States ≡ buckets.

& only those words are accepted, which are from a 'good' bucket.

(∵ for complementation; make good buckets bad, & bad buckets good).

Eg:



q_{00}: bucket with words which have even nos. of a's & b's

q_{10} :

q_{11} :

q_{01} :

now; since q_{10} is good,

L(G) = words with odd nos. of a's & even nos. of b's.

→ closure properties for regular language:-

•1) under complementation:-

mark good buckets as bad, and bad buckets as good.

L : $(Q, \Sigma, q_0, \delta, F)$

$\bar{L}$: $(Q, \Sigma, q_0, \delta, Q-F)$

•2) under intersection:-

G_1 ∩ G_2

$(Q_1, \Sigma, \delta_1, q_0, F_1)$ $(Q_2, \Sigma, \delta_2, s_0, F_2)$

new graph A

$(Q_1 \times Q_2, \Sigma, \delta, (q_0, s_0), F)$

cross product $= (q_1, q_2)$

> $\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$

> $F = F_1 \times F_2$ (both need to be good state).

we are capturing the parallel movement along both automatons.

naturally: $\hat{\delta}((q_1, q_2), w) = (\hat{\delta}_1(q_1, w), \hat{\delta}_2(q_2, w))$

•3) under union:-

G_1 ∪ G_2

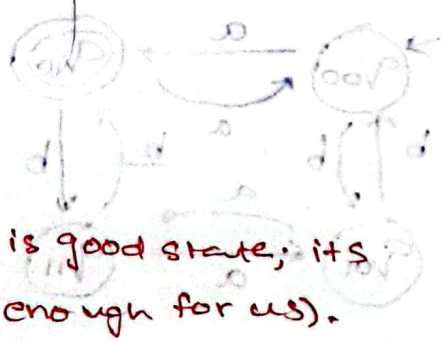
$(Q_1, \Sigma, \delta_1, q_0, F_1)$ $(Q_2, \Sigma, \delta_2, s_0, F_2)$

new graph A:

$(Q_1 \times Q_2, \Sigma, \delta, (q_0, s_0), F)$

> $\delta((q, s), a) = (\delta_1(q, a), \delta_2(s, a))$

> $F = (F_1 \times Q_2) \cup (F_2 \times Q_1)$ (if any one is good state, it's enough for us).



→ The comfort of Non-Determinism:-

we'll now look at a more relaxed model, which is as good as DFA.

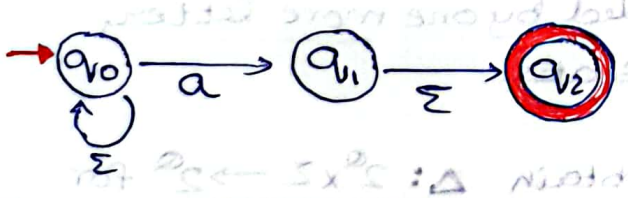
* we relax conditions on transition function δ;
  - $\delta: Q \times \Sigma \to 2^Q$ — can also have multiple start nodes.
  - $\delta(q_1, a)$ can be $\{q_0, q_1\}$. or even $\delta(q_1, a) = \phi$ is okay! ↓ interpret this as going to some trap state.

Eg:

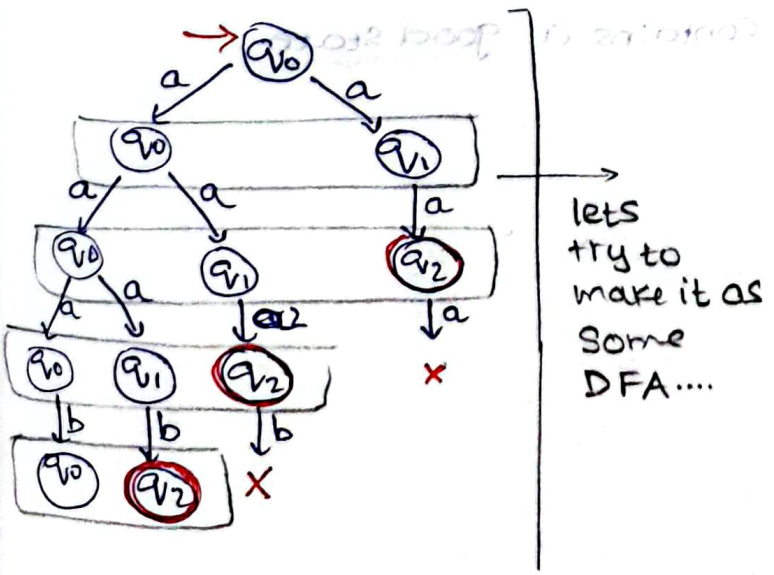
* is a word w accepatble by this NFA?

1. There'll be multiple runs of a single word 'w'.
2. For 'w' to be accepted should we impose
   all runs to lead to good states
   (or)
   Atleast one run to lead to good state. ← We'll take this. ✓

* for G:



run tree of word aaab is:-

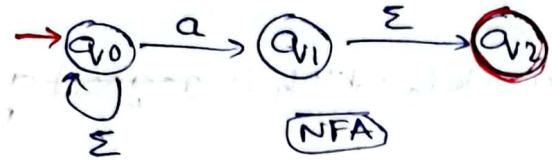


$\{q_0\}$ —a→ $\{q_0 q_1\}$ —a→ $\{q_0 q_1 q_2\}$ —a→ $\{q_0 q_1 q_2\}$ —b→ $\{q_1, q_2\}$

Something like quantum... we'll exist at many states at once.

final states in our new DFA, since; one of the states (q_2) is good state in NFA.

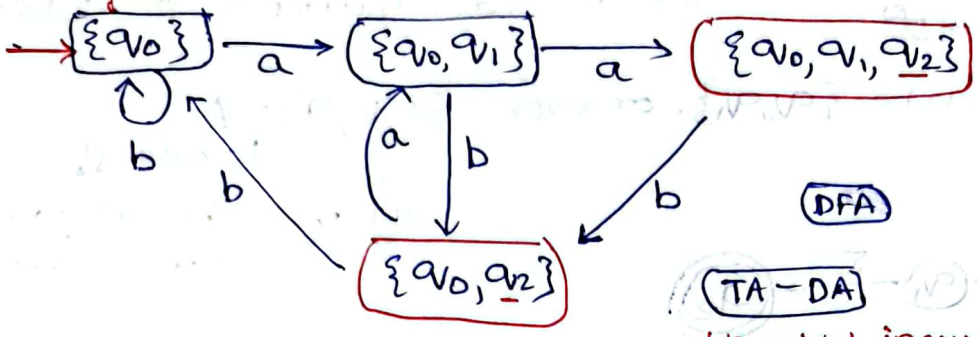
*



NFA

to

set of all start nodes.



DFA

TA – DA

We might incur exponential growth in no. of states.

---

* Any DFA is by default an NFA...
* Any NFA can be converted into a language preserving DFA.
  - combine all runs of 'w' in NFA, into a single run in DFA:
  - Combine all states occuring in various runs, to obtain a set of states.
  - A set of states; when forwarded by one more letter, gives rise to another set of states.
  - use $\delta: Q \times \Sigma \to 2^Q$ in NFA to obtain $\Delta: 2^Q \times \Sigma \to 2^Q$ for DFA.
  - Accept, if the (set of states) contains a good state.

↳ we can have multiple 'start nodes in NFA.

* Given NFA $N=(Q,\Sigma,Q_0,\delta,F)$, obtain the DFA $D=(2^Q,\Sigma,Q_0,\Delta,F')$

- $\Delta: 2^Q \times \Sigma \to 2^Q$ is defined as $\Delta(A,a) = \bigcup_{q \in A} \delta(q,a)$.
- $F' = \{S \in 2^Q \mid S \cap F \neq \phi\}$
- $\hat{\delta}(A,a) = \bigcup_{q \in A} \delta(q,a) = \Delta(A,a)$

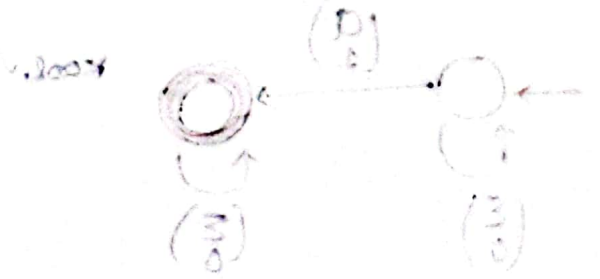
we can now show that both the above automatons have same language.

→ **Regularity of a language:-**

A language L is regular, iff there exists an NFA A, such that $L(A) = L$.

* Determinising an NFA can be (2^n) coststates finally.

∵ no. of subsets of Q $= 2^Q$.



→ Lets now start our pursuit of building an automaton for a FO formula φ:-

* Every FO sentence φ over words can be converted into a DFA A_φ such that $L(A_\varphi) = L(\varphi)$.
* Start construction from atomic formulae. Combine the DFAs of atomic formulae.
* conjunctions, disjunctions, negations → easily handled on DFAs.
* Quantifiers → yeaaa... handled too!

i) $Q_a(x)$: need to fix a position of x, where 'a' holds.

* baab satisfies $Q_a(x)$ with $x = 1$ or 2.

See as

b	a	a	b
0	1	0	0

b	a	a	b
0	0	1	0

i.e. new Alphabets are $\Sigma \times \{0,1\}$.

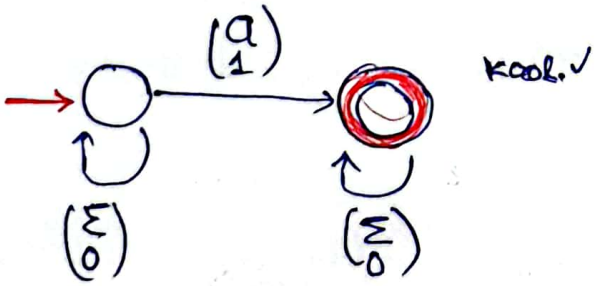
* first row is Σ; no constraints (to depict assignments — not only word structure)

2nd row is for x; only one value = 1, others == 0. (constraints).

↳ we NEED words that are constrained as such.

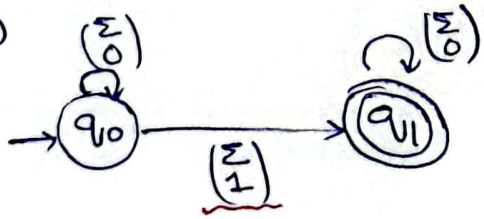
$\Sigma' : \Sigma \times \{0,1\}$

∴ $Q_a(x)$ would be:-



kaak. ✓

1.5) $(x = x)$



2) S(x,y):-

we've got two variables to assign now.

$$\Sigma' : \Sigma \times \{0,1\}^2$$

Eg: bab satisfies S(x,y) lol.

both structure and assignment!

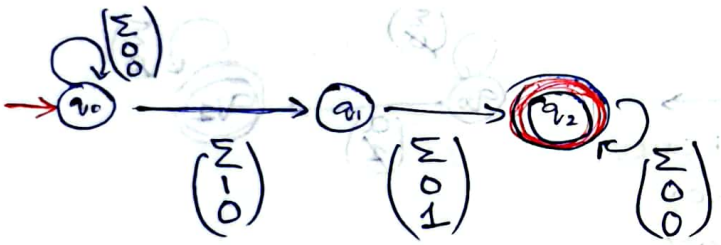
valid word is

b	a	b
1	0	0
0	1	0

b	a	b
0	1	0
0	0	1

for now, we don't have any quantifiers.

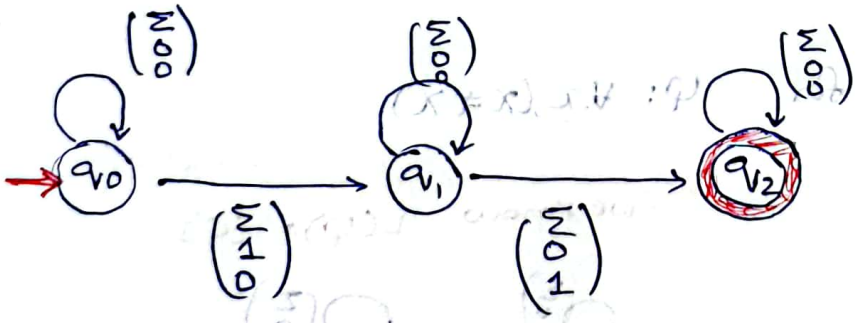
∴ S(x,y) would be :-



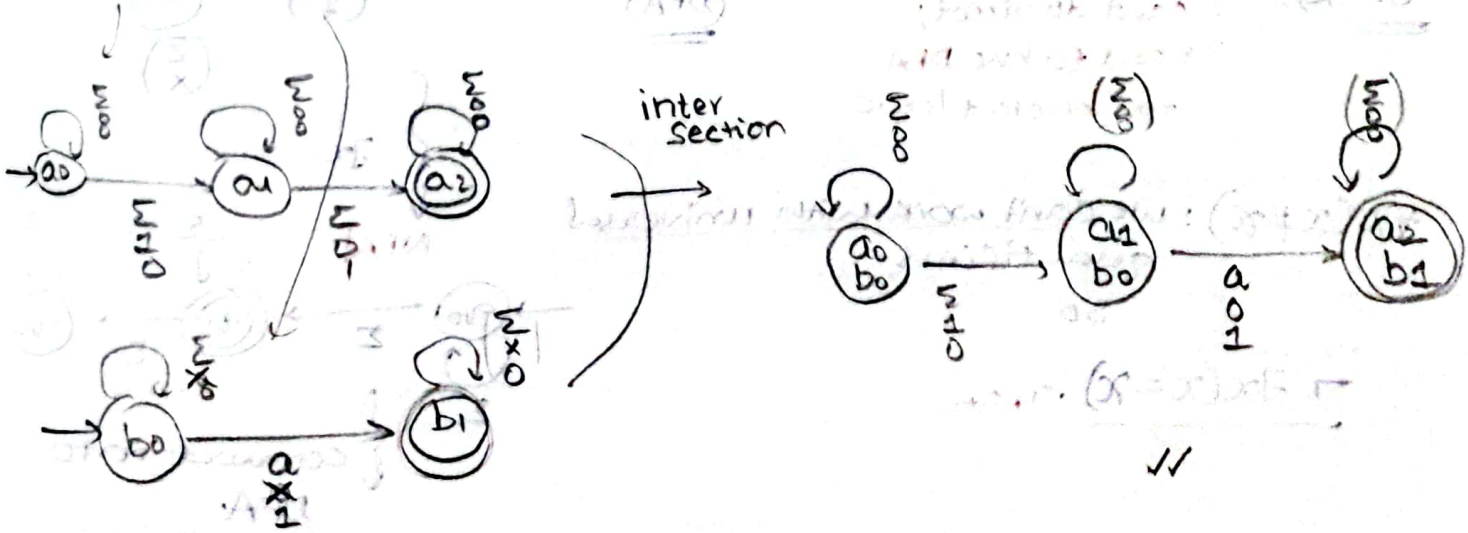
our constraints on 2nd, 3rd row still hold.

---

3) $<(x,y)$:-



Eg: $(x<y) \wedge Q_a(y)$ is automated as:-



//

* given $\varphi(x_1 x_2 \cdots x_n)$; an FO formula with 'n' free variables.

extended alphabet

$$\Sigma' : \Sigma \times \{0,1\}^n$$

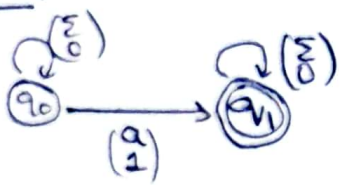
constraint:- every row, except the first, can be 1 at unique position

---

→ Showing quantifiers:-

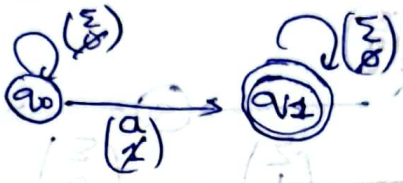
* remember! $Q(x)$; $\exists x Q(x)$ are equi-SAT.

$Q_a(x)$:



project over 1st row; So ignore 2nd row values.

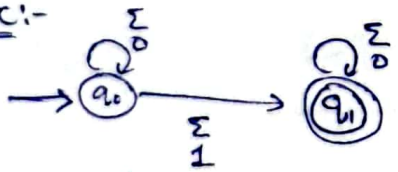
$\exists x Q_a(x)$:



This'll convert DFA to NFA

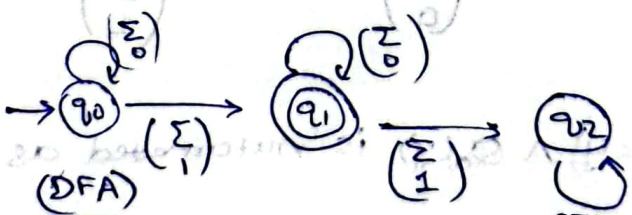
Q: draw automaton for $\varphi: \forall x (x \neq x)$

$x = x$:-



$\neg(x = x)$:- { can't do direct; need to have DFA ( ) for bucket logic

we know $L(\varphi) = \{\in\}$.



(DFA)

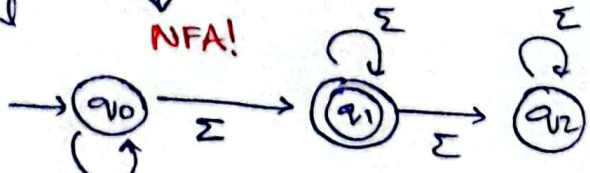
$\forall x (x \neq x)$: We can't work with universal quantifiers.

So

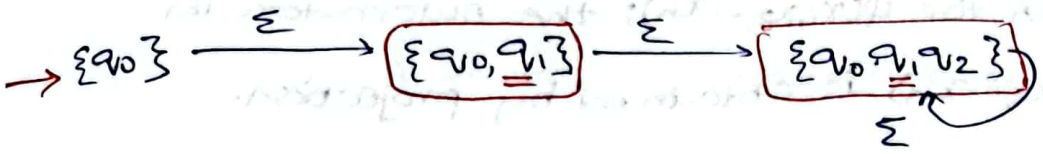
$\neg \exists x (x = x)$. TADA

$\exists x.$

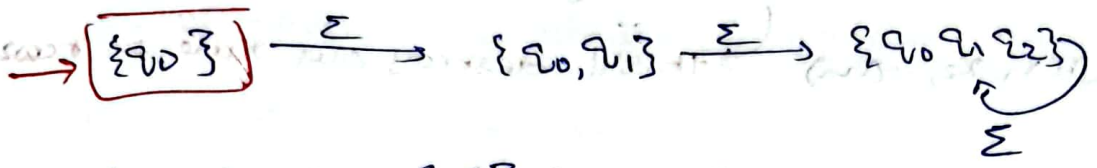
NFA!



convert into DFA.



∴ ¬∃x(x=x) will be:



∴ only the {∈} is our language.

* Summary:-

* let $L \subseteq (\Sigma \times \{0,1\}^n)^*$ be defined by $\varphi(x_1 x_2 \cdots x_n)$.

* let $f: (\Sigma \times \{0,1\}^n)^* \to (\Sigma \times \{0,1\}^{n-1})^*$ be the projection

$f(w, c_1, c_2, \cdots c_n) = (w, c_1, c_2, \cdots c_{n-1})$

then

$\exists x_n \varphi(x_1, x_2, \cdots x_{n-1})$ defines $f(L)$.

projection dude!

$\forall x_n \neg\varphi(x_1 x_2 \cdots x_n)$ is done as $\neg\exists x_n \varphi(x_1 x_2 \cdots x_n)$

∈ ('¬') can be done, only on DFAs. NOT on NFA.

* Also; An issue crops up:-

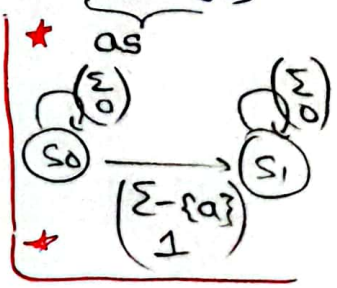
$\exists y \forall x [x \leq y \wedge Q_a(y)]$ is done as $\neg\exists x [x > y \vee \neg Q_a(y)]$

* when we finally do

¬(-DFA-).

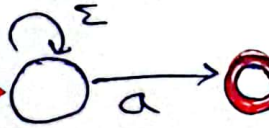
- we'll notice our constraint on 2nd row (i.e. only 1 place is 1, others = 0) doesn't hold.

- Hence; conjunction with $\binom{\Sigma}{0}^* \binom{\Sigma}{1} \binom{\Sigma}{0}^*$.



Since; complement means; Superset is $\binom{\Sigma}{x}^*$.

final answer:



(; could have drawn directly).

* Given the DFA for $\varphi(x_1, x_2, \cdots x_n)$; the Automaton for $\exists x_1 \varphi(x_1, x_2, \cdots x_n)$ is obtained by projection.

→ May get an NFA.

→ 2^n cost.

* Determinise it & complement to get DFA for

$\neg \exists x_1 \varphi(x_1, x_2, \cdots x_n)$ → this'll destroy our rule on rows 2, 3, ...

* Intersect with regular language 'n-1' times where each of $x_1, x_2, \cdots x_{n-1}$ will occur ~~atmost~~ exactly 1 time.

$$\left(\begin{matrix}\Sigma\\ x\\ x\\ 0\\ x\\ \vdots\end{matrix}\right)^* \left(\begin{matrix}\Sigma\\ x\\ x\\ 1\\ x\\ \vdots\end{matrix}\right) \left(\begin{matrix}\Sigma\\ x\\ x\\ 0\\ x\\ \vdots\end{matrix}\right)^*$$

* complementing, when we have alphabet : $\Sigma \times \{0,1\}^n$; will break our rule of "only one 1".

Bcoz; the whole set would be $\left(\begin{matrix}\Sigma\\ x\\ x\end{matrix}\right)^*$.

→ computational effort :- Growth in size of automaton.

* Given 2 NFAs A_1, A_2 each with almost n states :-

union → $2n$ states. (∵ write both side by side). awesome!

intersection → n^2.

complement → 2^n states (∵ have to determinise first)

projection → n states.

Eg: $\exists x_1 \forall x_2 \exists x_3 \forall x_4 \cdots \varphi$; now each '∀' quantifier'll have exponential blow up.

$$\text{Automaton} = 2^{2^{\cdots^{2^n}}}$$

where tower height = no. of times $(\exists\forall)$ occured.

∴ I claim;

> Given an FO sentence φ; one can construct a DFA A_φ such that $L(\varphi) = L(A_\varphi)$.

* When I want to say last of a word follows a property F; I can write in formula as:-

$\exists x (\forall y (y \leq x) \wedge Q_a(x))$

$\forall x (\forall y (y \leq x) \rightarrow Q_a(x))$

better. ✓ { $\exists x (last(x) \wedge F)$ ✓

$\forall x (last(x) \rightarrow F)$ ✓

works; cuz only one element satisfies last(x).

L8.Pdf:- Monadic Second Order Logic (MSO):-

* Formulas of MSO; over signature τ, are sequence of symbols where each symbol is one of the following:- (just writing the minimal set).
  - The symbol '$\perp$' called false
  - an element of the infinite set $\mathcal{V}_1 = \{x_1, x_2 \cdots\}$ of first-order variables
  - an element of the infinite set $\mathcal{V}_2 = \{X_1, X_2 \cdots\}$ of second-order variables
  - constants and relations from τ.
  - $\rightarrow$; $\forall$; ( ) parenthesis.

structure A definition is same as before

structure doesn't deal with variables. assignment does

* well-formed formulae; over signature τ are inductively defined as follows
  - $\perp$ is a wff
  - if t_1, t_2 are constants/variables; then $t_1 = t_2$ is wff.
  - if t_i is constant/FO variable, & R is a K-ary relation in τ; $R(t_1, t_2, \cdots t_k)$ is wff.
  - if t is constant/First-order variable, X is second-order var; $X(t)$ is wff.
    arity = 1; hence monadic.
  - if ψ, φ are wff, $\varphi \rightarrow \psi$ is a wff.
  - if φ is wff; $(\forall x)\varphi$ is wff. x is FO variable.
  - if φ is wff; X is second order var; $(\forall X)\varphi$ is wff.

* Sentence is a MSO formula; with no free FO/SO variables.

→ Assignments on τ-structures:-

* For a τ-structure $\mathcal{A}$; assignment α is a pair of functions (α_1, α_2) where

$\alpha_1 : \mathcal{V}_1 \longrightarrow U(\mathcal{A})$

$\alpha_2 : \mathcal{V}_2 \longrightarrow 2^{U(\mathcal{A})}$; assigns to every SO var; a subset of $U(\mathcal{A})$

in automatons, that rule of "only one 1" is not needed anymore...

* $\mathcal{A} \models_\alpha \varphi$ is defined as:-

; the usual; for FO too!

- $\mathcal{A} \models_\alpha X(t)$ iff $\alpha_1(t) \in \alpha_2(X)$.
- $\mathcal{A} \models_\alpha (\forall X)\varphi$ iff for every $S \subseteq U(\mathcal{A})$; $\mathcal{A} \models_{\alpha[X \to S]} \varphi$

every subset of universe rather than every element. ($\forall x \varphi$).

Eg:- Graph Structure... $\tau = \{E\}$

1. Graph is 3 colorable:-

$$\exists X, Y, Z \Big(\forall x \big(X(x) \vee Y(x) \vee Z(x)\big) \wedge \forall x \forall y \big[E(x,y) \to \{\neg(X(x) \wedge X(y)) \wedge \neg(Y(x) \wedge Y(y)) \wedge \neg(Z(x) \wedge Z(y))\}\big] \Big)$$

each stands for a color.

2. Graph of independent set $\geq k$:-

$$\exists I \{ \forall x, y \big((\neg(x=y)) \wedge I(x) \wedge I(y) \big) \to \neg E(x,y) \big)$$

$$\wedge \exists x_1 x_2 \cdots x_k \Big[\bigwedge_{i \neq j} \neg(x_i = x_j) \wedge \bigwedge_i I(x_i) \Big] \}$$

okay...

* $\Sigma = \{a, b\}$

L = words of even length.

L is not FO-definable

L is regular.

wat about MSO?] we'll see Regular = MSO.

$$\exists E, O \{ \forall x [(first(x) \rightarrow E(x)) \wedge (last(x) \rightarrow O(x))]$$

$$\wedge \forall x [(E(x) \vee O(x)) \wedge \neg (E(x) \wedge O(x))]$$

$$\wedge \forall x, y [S(x,y) \wedge O(x) \rightarrow E(y)]$$

$$\wedge \forall x, y [S(x,y) \wedge E(x) \rightarrow O(y)] \}$$

→ MSO on words:-

* $\mathcal{T} = (Q_\Sigma, <, S)$, domain / universe = set of positions of word.

* Atomic formulae in MSO over words are

$$X(x) \mid Q_\Sigma(x) \mid x = y \mid X = Y \mid x < y \mid S(x,y)$$

* Given an MSO sentence; we'll define $L(\varphi)$ as before.

* "SO definable or MSO-definable language".

* Lets asks it now;

Satisfiability of MSO sentence.

* clearly FO ⊆ MSO

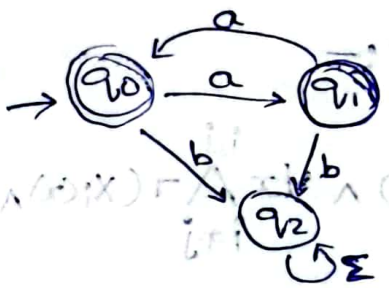
we know FO ⊆ REG

∴ REG ≡ MSO lol. but true!

aSo we'll see it.

→ Regular language to MSO:-

Eg:



run for the word aaba

a a b a
q_0 q_1 q_0 q_2 q_2

each state is a SO-variable...

works only for DFA??

I think NFA too okay.....

Here

$X_{q_0} = \{0, 2\}$

$X_{q_1} = \{1\}$

$X_{q_2} = \{3\}$

& ending is q_2.

initial position always to q_0

* Show a state q_i using a SO-variable X_{q_i}. we are trying to capture what is happening in DFA.

- if x, y are consecutive positions;

$$X_{q_i}(x) \wedge Q_a(x) \longrightarrow X_t(y)$$

where $t = \delta(q_i, a)$

- say w_a is in Language.

length = k+1.

then $[Q_a(k) \wedge X_{q_i}(k)]$ such that $\delta(q_i, a) \in F$.

∴ we'll keep in formula,

$$\bigvee_{\delta(i,a)=j \in F \text{ over } i,a} \left[X_i(last) \wedge Q_a(last) \right]$$

Now; lets complete our discussion; PTO!!!

* Given a DFA $A = (Q, \Sigma, \delta, q_0, F)$ a word 'w' is accepted iff it satisfies the MSO-formula φ :-

$$\exists X_0, X_1, \cdots X_n \Big\{ \underbrace{\Big[\forall x (X_0(x) \vee X_1(x) \vee \cdots X_n(x)) \wedge \forall x \bigwedge_{i \neq j}^{i,j} \neg (X_i(x) \wedge X_j(x)) \Big]}_{\text{Basic}}$$

$$\wedge$$

$$\underbrace{\exists x [first(x) \wedge X_0(x)]}_{\text{first start from state 0}}$$

$$\wedge$$

$$\underbrace{\forall x \forall y \Big[S(x,y) \longrightarrow \bigvee_{\delta(i,a)=j}^{i,a} [X_i(x) \wedge Q_a(x) \wedge X_j(y)] \Big]}_{\text{transition step.}}$$

$$\wedge$$

$$\underbrace{\exists x \Big[last(x) \wedge \bigvee_{\delta(i,a)=j \in F}^{i,a} [X_i(x) \wedge Q_a(x)] \Big]}_{\text{Last state is a good state....}} \Big\}$$

* This ( ) doesn't cover the possibility of empty word. woah! sooo huge!

So; if state '0' $\in F$; write $\varphi' = \varphi \wedge \forall x (x \neq x)$.

for completeness sake.

$\therefore$ every DFA can be converted to MSO-formula.

→ MSO to DFA:-

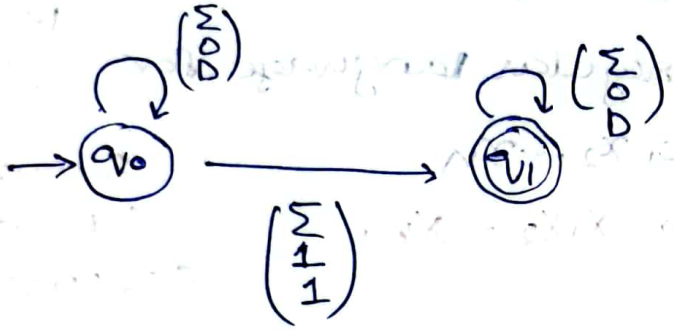
* Every MSO formula can be converted into a DFA, Ap. (over words)

* Start with the atomic formulae!

  we've done for all before!.

  only $X(x) \cdot \subseteq$, $\exists X$ ---- this also just projection.

→ $X(x)$:- two variables X, x. two extra rows.

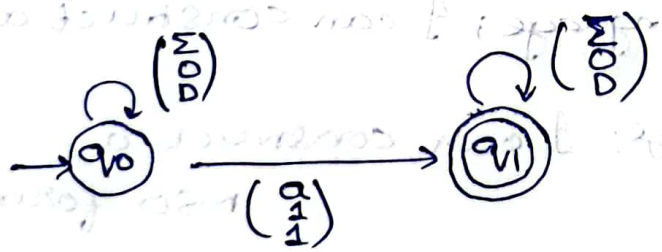


∴ Here; Row 1 for x; "only one 1!"

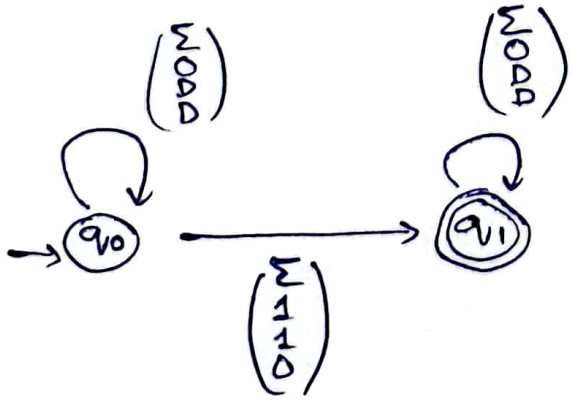
Row 2 ⟶ no such restriction!

D → dontcare.

* $Q_a(x) \wedge X(x)$:-



* $X(x) \wedge \neg Y(x)$:-



* rules → Same as before.

if n x_is
m X_js then $\Sigma \times \{0,1\}^{m+n}$.

∃ rule of projecting ✓

¬ rule :- Should have a DFA.

& after projecting, complementing, we need to intersect with regular language for variables $x_1 x_2 \cdots x_n$.

need not for $X_1 X_2 \cdots X_n$.

I told ya why.

Because our superset $(\Sigma \times \{0,1\}^n)$* contains non-regular too! wrt x_i.

→ The Automaton-logic connection:-

* MSO ≡ Regular

* for a MSO-definable language; I can construct an NFA;

for a regular language; I can construct a MSO-formula.

cheat sheet:-

P⊕Q as $(P \vee Q) \wedge (\neg P \vee \neg Q)$ (POS form) | HornSat: worst condition → quadratic- Xor-Sat → elimination.

1) from prev sheets,
2) substitution thms
3) the conclusions for basic formulae...

Proof System:-

- Assumption $\frac{}{\Sigma \vdash F}\ F \in \Sigma$
- Monotonic $\frac{\Sigma \vdash F}{\Sigma' \vdash F}\ \Sigma \subseteq \Sigma'$
- DoubleNeg $\frac{\Sigma \vdash F}{\Sigma \vdash \neg\neg F}$
- RevDoubleNeg $\frac{\Sigma \vdash \neg\neg F}{\Sigma \vdash F}$
- ∧-Intro $\frac{\Sigma \vdash F \quad \Sigma \vdash G}{\Sigma \vdash F \wedge G}$
- ∧-Elim $\frac{\Sigma \vdash F \wedge G}{\Sigma \vdash F}$ (don't write G; use ∧-SYMM)
- ∧-PAREN $\frac{\Sigma \vdash (A \wedge B) \wedge C}{\Sigma \vdash A \wedge B \wedge C}$
- V-INTRO $\frac{\Sigma \vdash F}{\Sigma \vdash F \vee G}$
- V-SYMM $\frac{\Sigma \vdash F \vee G}{\Sigma \vdash G \vee F}$
- V-DEF $\frac{\Sigma \vdash F \vee G}{\Sigma \vdash \neg(\neg F \wedge \neg G)}$ and vice versa.
- V-ELIM $\frac{\Sigma \cup F \vdash H \quad \Sigma \cup G \vdash H}{\Sigma \vdash H}$ useful!
- ⇒-INTRO $\frac{\Sigma \cup F \vdash G}{\Sigma \vdash F \Rightarrow G}$ Nice. not viceversa. use ⇒elim. ...
- ⇒-ELIM $\frac{\Sigma \vdash F \Rightarrow G \quad \Sigma \vdash F}{\Sigma \vdash G}$
- ⇒-DEF $\frac{\Sigma \vdash F \Rightarrow G}{\Sigma \vdash \neg F \vee G}$ and viceversa
- ⇔-DEF $\frac{\Sigma \vdash F \Leftrightarrow G}{\Sigma \vdash F \Rightarrow G,\ \Sigma \vdash G \Rightarrow F}$ and viceversa.

- Modusponens $\frac{\Sigma \vdash \neg F \vee G, \Sigma \vdash F}{\Sigma \vdash G}$
- Tautology $\frac{}{\Sigma \vdash F \vee \neg F}$
- Contradiction $\frac{\Sigma \vdash F \wedge \neg F}{\Sigma \vdash G}$

used in places, where say... $\Sigma \cup \{\neg F\} \vdash \bot$; just to preserve notion.

- Contrapositive $\frac{\Sigma \cup F \vdash G}{\Sigma \cup (G) \vdash \neg F}$
- By Cases $\frac{\Sigma \cup F \vdash G, \Sigma \cup \neg F \vdash G}{\Sigma \vdash G}$
- By Contradiction $\frac{\Sigma \cup F \vdash G, \Sigma \cup F \vdash \neg G}{\Sigma \vdash \neg F}$ usually to prove tricky but trivial results....
- Resolution $\frac{\Sigma \vdash F \vee G \quad \Sigma \vdash \neg F \vee H}{\Sigma \vdash G \vee H}$

FO Proof System:-

- ∃-INTRO $\frac{\Sigma \vdash F(t)}{\Sigma \vdash \exists y F(y)}$
- ∀-INTRO $\frac{\Sigma \vdash F(x)}{\Sigma \vdash \forall x F(x)}$ No reference condition.
- ∀-Elim $\frac{\Sigma \vdash \forall x F(x)}{\Sigma \vdash F(t)}$ ∀-instantiation
- ∃-Elim $\frac{\Sigma \vdash F(x) \Rightarrow G}{\Sigma \vdash \exists x F(x) \Rightarrow G}$ ∃-instantiation; No reference condition.
- Reflex $\frac{}{\Sigma \vdash t = t}$
- EQSUB $\frac{\Sigma \vdash F(t) \quad \Sigma \vdash t = s}{\Sigma \vdash F(s)}$
- EQSYM $\frac{\Sigma \vdash s = t}{\Sigma \vdash t = s}$

Substitution theorem:-

F(p), H, G, model m

if $m \models G$ iff $m \models H$; then

$m \models F(G)$ iff $m \models F(H)$.

Equivalence generalisation theorem:-

if $F(p) \equiv G(p)$; then for each H; $F(H) \equiv G(H)$

Subformula replacement:-

F(G) ≡ G, H, F(p).

if $G \equiv H$ then $F(G) \equiv F(H)$.

MGU:- (t, u)

σ := {}.
while t ≠ u do
  d_1, d_2 be in t, u
  if both are not vars, fail
  if d_1 ∈ vars then
    $x = d_1, s = d_2$
  if d_2 ∈ vars
    $x = d_2, s = d_1$
  if $x \in FV(s)$ then fail.
  $\sigma := \sigma[x \to s]$

* in completeness of fol:-
  $res^0(\Sigma) = \Sigma$
  $res^1(\Sigma)$ = one time applied
  Σ_0 = clauses with p; Σ_0' → delete p.
  Σ_1 = clauses with ¬p; Σ_1' → delete p.
  Σ^* = remaining.

* theorem on strings: there exist a infinite word w; prefix w is path of no way on S.

0*1 means 01, 001, 0001, ...

* FO can't capture finity. domain

For resolution:- only on FO-sentences. $\Phi_1 \wedge \Phi_2 \cdots$ is implicit.

- Same var of same clause → consider $x_1\ x_1$
- Same var of different clause → consider $x_1\ x_2$.

} Helps when $\neg R(x) \wedge R(f(x))$. (x_1, x_2)

MGU = $x_1 \to f(x)$

resln done!

- Resolution $\dfrac{\neg A \vee C \quad B \vee D}{(C \vee D)\sigma}$ $\sigma = mgu(A,B)$
- FACTOR $\dfrac{L_1 \vee L_2 \vee L_3 \cdots \vee L_n}{(L_1 \vee C)\sigma}$ $\sigma = mgu(L_1, L_2 \cdots L_n)$

} used when $\Sigma \vdash P(x) \vee P(y)$

put $\sigma = [x \to y]$.

$\therefore \Sigma \vdash P(x) \vee \cdots$

equality

- PARAMODULATION $\dfrac{s = t \vee C \quad D[u]}{(C \vee D[t])\sigma}$ $\sigma = mgu(s,u)$

inequality:

- Reflexivity $\dfrac{t \neq u \vee C}{C\sigma}$ $\sigma = mgu(u,t)$

---

* Not regular: $L = \{a^n b^n \mid n \geq 1\}$
- Bucket interpretation.
- NFA to DFA.

Costs:-

union $2n$
intersect n^2
complement 2^n
projection n.

intersect with regular language over always; whenever complement.

First(x):-

$\exists x \, (\forall y \; x \leq y)$

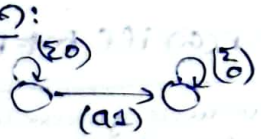
Last(x):-

$\exists x \forall y \; [y \leq x \wedge \text{property on } x]$

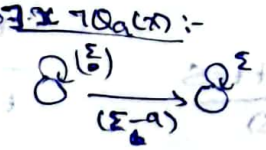
$\exists x_1 \forall x_2 \exists x_3 \forall x_4 \cdots$ — n alternants

$2^{2^{2^{\cdots^{2^n}}}}$ (tower) — n height

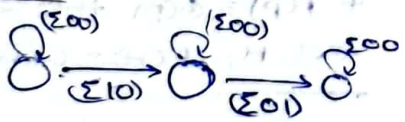
$Q_a(x)$:



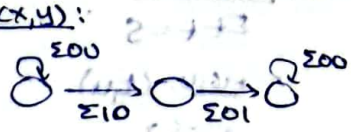
$\exists x \, \neg Q_a(x)$:-



$x < y$:

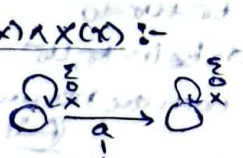


$S(x,y)$:



$X(x)$:

$Q_a(x) \wedge X(x)$:-



DFA to MSO:-

$A = (Q, \Sigma, \delta, q_0, F)$

ψ is

$$\exists X_0, X_1, \cdots X_n \Big\{ \forall x (X_0(x) \vee X_1(x) \cdots X_n(x)) \wedge \forall x \bigwedge_{i \neq j} \neg(X_i(x) \wedge X_j(x)) \wedge \exists x \forall y [x \leq y \wedge X_0(x)] \wedge \forall x \forall y \Big[S(x,y) \to \bigvee_{\delta(i,a)=j} (X_i(x) \wedge Q_a(x) \wedge X_j(y)) \Big] \wedge \exists x \forall y \Big[y \leq x \wedge \bigvee_{\delta(i,a)=j \in F} (X_i(x) \wedge Q_a(x)) \Big]$$

doesn't cover empty word.

$\therefore$ if $X_0 \in F$; add

$\forall x \, (X \neq X)$ as disjunct

$x.b$ encoded as

$\exists y \, (S(x,y) \wedge Q_b(y))$ } compulsory, successor is b.

$\forall y \, (S(x,y) \Rightarrow Q_b(y))$ } if successor exists; it's b.