# LECTURE 4
## LOGIC PROGRAMMING
## PROPOSITIONAL LOGIC - COMPLETION

So far we considered **brute force** technique of showing

$$\boxed{\Delta \models Q} \qquad (\blacktriangle)$$

based on **truth table** (based on definition of $(\blacktriangle)$).

Since **RESOLUTION is SOUND** for $\Delta$ & $Q$ being clauses we have an **alternative:**

**FORWARD CHAINING INFERENCE**
FCI

1. Start with an initial set of clauses $\Delta$.
2. Choose any 2 clauses that can be resolved. Obtain the resolvent and add it to $S$ i.e.

$$\hat{S} = S \cup \{\text{Resolvent}\}$$
$$\| \Delta$$

3. Such process should be repeated until no new clauses can be obtained (or until we reach Q). Elements of the resulting set $S^*$ (called **RESOLUTION CLOSURE**) are then logical consequences of the original set $S = \Delta$.

One can now retranslate the definitions of **SOUNDNESS** & **COMPLETENESS** for RESOLUTION, $\Delta$ set of clauses & Q a queried clause. Namely:

a) $\boxed{\text{if } Q \in S^* \Rightarrow S \vdash Q}$ ?

soundness

b) $\boxed{\text{if } S \vdash Q \Rightarrow Q \in S^*}$ ?

completeness

# LECTURE 4
## LOGIC PROGRAMMING

**Example 1:** Forward chaining inference.

Let $\Delta = S = \{ p;\ q;\ r;\ w \leftarrow p, r;\ v \leftarrow w, q, s;\ s \leftarrow w \}$

$\mathcal{J} = \{ \text{Resolution} \}$

$S \leftarrow$ consists of clauses.

**Question:** $\boxed{S \models v}$ ?

**Method 1:** truth table

$\Delta_a = \{ p, q, r, w, v, s \}$

Number of all interpretations (so possible models for $\Delta$) is $2^{\underset{=}{\Delta_a}} = 2^6 = 64$.

Clearly checking all of them is a bit awkward.

**Method 2:** forward chaining inference.

$\tilde{S} := S$

**1° Step:**

$$\frac{\overset{p}{w \leftarrow p, r}}{w \leftarrow r}$$ Resolvent 1

$\tilde{S} := \tilde{S} \cup \{ w \leftarrow r \}$.

2. Step:

$$\frac{\overset{r}{\omega \leftarrow r}}{\omega}$$ Resolvent_2

$\tilde{S} := \tilde{S} \cup \{\omega\}$

3. Step:

$$\frac{\overset{\omega}{v \leftarrow \omega, q, s}}{v \leftarrow q, s}$$ Resolvent_3

$\tilde{S} := \tilde{S} \cup \{v \leftarrow q, s\}$

4. Step:

$$\frac{\overset{q}{v \leftarrow q, s}}{v \leftarrow s}$$ Resolvent_4

$\tilde{S} := \tilde{S} \cup \{v \leftarrow s\}$

5. Step:

$$\frac{\overset{\omega}{s \leftarrow \omega}}{s}$$ Resolvent_5

$\tilde{S} := \tilde{S} \cup \{s\}$

6. Step:

$$\frac{\overset{s}{v \leftarrow s}}{\blacksquare}$$ Resolvent_6

So after 6 steps of RESOLUTIONS we arrive at v. By **Soundness** of **Resolution** we have:

$$\boxed{S \models v}$$ yes.

□

**Forward chaining inference:**

- requires the specific choices of clauses.
- Q is to be a clause
- the hope is the computer will do this job for us.

**PROLOG** uses however:

- neither **truth table** technique
- nor **forward chaining inference**

to prove $\boxed{\Delta \models Q}$

# LECTURE 4
## LOGIC PROGRAMMING

**Method 3:**

is called **REFUTATION** which still uses RESOLUTION but in a modified fashion.

The reason why **REFUTATION** is used:

- the proof for completeness & soundness of RESOLUTION is based on REFUTATION proving that $A \models Q$.
- there is also another reason which will be clear once we pass to PREDICATE LOGIC (explained later)

We introduce first some auxilliary notions:

- satisfiability
- unsatisfiability

# LECTURE 4.
## LOGIC PROGRAMMING

**DEFINITION 1:** A set of formulas $S$ is **satisfiable** if there is at least one interpretation $I$ which is a model for $S$ i.e. $\forall f \in S \quad I(f) = \text{true}$.
Otherwise $S$ is called **unsatisfiable**.

**Example 2:** **a)** $S = \{\sim p ; p\}$ is unsatisfiable.

There is no model for $S$.

**b)** $S = \{p ; q\}$ it is satisfiable

there is one model for $S$ $I(p) = I(q) = \text{true}$

The remaining 3 interpretations:

$\Delta_a = \{p, q\}$ $2^2 = 4$

$I_1(p) = I_1(q) = \text{false}$

$I_2(p) = \text{true}$ & $I_2(q) = \text{false}$

$I_3(p) = \text{false}$ & $I_3(q) = \text{true}$

are not models for $S$. ■

Note that if all interpretations are models for $S$ then we say that any $f \in S$ is

**A TAUTOLOGY**

# LECTURE 4
## LOGIC PROGRAMMING

**Example 3:** Tautologies e.g.

$\sim(p \wedge q) \equiv \sim p \vee \sim q$
$\sim(p \vee q) \equiv \sim p \wedge \sim q$
} de Morgan laws

$p \vee \sim p$
⋮

Symbol "$\equiv$" means $L \equiv R$

$L \leftarrow R$ & $R \leftarrow L$.

∎

**Theorem 1:** Let S be a set of legal formulas. Then

$S \models Q$ if and only if (iff)

iff

$S \cup \{\sim Q\}$ is unsatisfiable

**PROOF:**

(i) Assume $S \models Q$ $\overset{?}{\Rightarrow}$ $S \cup \{\sim Q\}$ unsatisfiable

a) if there is no model for S then there is no model for $S \cup \{\sim Q\}$.
So here $S \cup \{\sim Q\}$ is unsatisfiable.

b) take arbitrary model for S. under I since $S \models Q$ then $I(Q) = \text{true}$.

Thus $I(\sim Q) = \text{false}$.

It i.e. I cannot be now a model for $S \cup \{\sim Q\}$ since $I(\sim Q) = \text{false}$.

So $S \cup \{\sim Q\}$ is unsatisfiable.

c) if we take an interpretation I which is not a model for S

$\Rightarrow$ I cannot be a model for $S \cup \{\sim Q\}$ as $\exists f \in S$ such that $I(f) = \text{false}$.

Hence $S \cup \{\sim Q\}$ is unsatisfiable.

All cases are covered.

Thus $S \cup \{\sim Q\}$ is unsatisfiable.

(ii) Assume $S \cup \{\sim Q\}$ is unsatisfiable.

$\Downarrow$ ?

$S \models Q$

(a) if there is no model for $\Delta$ then by def. of logical consequences

$$S \models Q.$$

(b) take arbitrary model I for S.

We need to show that $\boxed{I(Q) = \text{true}}$ ?

Since $S \cup \{\sim Q\}$ is unsatisfiable

$\exists\ \tilde{f} \in S \cup \{\sim Q\}$

such that $I(\tilde{f}) = \text{false}$

but $\forall f \in S \quad I(f) = \text{true}$ (I is a model for S)

Hence $I(\sim Q) = \text{false}$ (otherwise $S \cup \{\sim Q\}$ is satisfiabe)

$$\Downarrow$$

$$\boxed{I(Q) = \text{true}}$$

Thus $\boxed{S \models Q.}$

The proof is complete. ■

# LECTURE 4
## LOGIC PROGRAMMING

**Method 3:**

of showing $\Delta \models Q$

a) add negation of QUERY Q to $\Delta$:

i.e. $\tilde{\Delta} = \Delta \cup \{\sim Q\}$

b) prove that:

$\tilde{\Delta}$ is unsatisfiable

Note: that Theorem 1 is valid for arbitrary set of legal formulas $\Delta$. It is also (as shown later) naturally extendable to $\Delta$ in PREDICATE LOGIC.

If $\Delta$ is **set of clauses** (for PROLOG)
↑ specially formatted set of formulas

then:

# LECTURE 4
## LOGIC PROGRAMMING

for RESOLUTION to act

$$\tilde{\Delta} \equiv \Delta \cup \{\sim Q\}$$ ≡ set of clauses

$\Rightarrow$ $\sim Q$ has to be a clause!

So for REFUTATION METHOD (in PROPOSITIONAL LOGIC) by deMorgan law:

$$Q \equiv p_1 \wedge \ldots \wedge p_n \quad (\bullet)$$

conjunction (not a clause)

$$\sim Q \equiv \sim p_1 \vee \sim p_2 \vee \ldots \vee \sim p_n$$

a clause (a negative clause).

**1. PROGRAMMER:**

(i) keeps a program $\Delta$ with the non-negative clauses of type

either $A \leftarrow$ facts

or $A \leftarrow B_1, B_2, \ldots B_n$ rules

(ii) loads a program to PROLOG data base

(iii) queries PROLOG whether Q (in a format (•))

$$\Delta \models Q$$

2. PROLOG interpreter:

verifies by "magic activity"

whether

$$\Delta \models Q \quad \text{or} \quad \Delta \not\models Q.$$

"Magic activity" based on REFUTATION will be explained later within more general setting of PREDICATE LOGIC.

For next examples:

a) read more 60 lecture extract 2.6. paragraph.

b) manual to SWI 3.2.9 - WEB access

# LECTURE 4
## LOGIC PROGRAMMING

**Example 4:** PROLOG program for Example 1

$\Delta \models v$ ?

$\Delta$ = example4.pl

```
p.
q.
r.
w:-p,r.
v:-w,q,s.
s:-w.
```

LINUX SHELL

1. >
2. pl (+RTN) Calling PROLOG SESSION
3. >> PROLOG SESSION opens
4. >> consult('example4.pl'). (+RTN)
5. >> v. (+RTN) Querying whether $\Delta \models v$.
6. >> yes PROLOG replies yes.
7. >> PROLOG waits for more QUERIES
8. >> halt. (+RTN) leaving PROLOG SESSION return to LINUX SHELL.
9. >

# LECTURE 4
## LOGIC PROGRAMMING

**Example 5:**

Student took various courses.

He/she wants to check whether he satisfied maths requirements for degree.

It is satisfied if he did:

1) calculus sequence
2) finite maths sequence
3) algebra sequence

In PROLOG 1) 2) & 3) are:

```
mathReq:- calcSeq, discreteSeq, algSeq.
```

calSeq: is true if he/she did it

discreteSeq: is true if he/she did it

algSeq: is true if he/she did it

# LECTURE 4
## LOGIC PROGRAMMING

Each of these sequences requires certain courses:

II
```
calSeq :- basicCalc, advCalc.
```

indicates that calculus sequence requires basic & advance calculus.

Basic calculus can be satisfied in three ways:

III
```
basicCal :- takenCalc1, takenCalc2.
basicCal :- takenCalcA, takenCalcB.
basicCal :- schoolCalc.
```

by either taking Calc1 & Calc2
or taking CalcA & CalcB
or getting exemption based on school calculus school.

Note <u>OR</u> is expressed in <u>separate</u> clauses, AND in adding literals to a given clause.

# LECTURE 4
## LOGIC PROGRAMMING

Advance calculus is just one further course:

IV
```
advCal :- takenM222.
```

Discrete sequence requires one of graph theory & combinatorics or one of those together with numerical analysis:

V
```
discreteSeq :- takenGraph, takenComb.
discreteSeq :- takenGraph, takenNumA.
discreteSeq :- takenComb , takenNumA.
```

Finally the algebra requirements are linear algebra & group theory

VI
```
algSeq :- takenLin, takenGroup.
```

So far we put **Rules** (conditions).
I-VI
Now for a particular student we can enter the course he/she did.

# LECTURE 4
# LOGIC PROGRAMMING

E.g.

$\overline{VII}$

```
takenGraph.
takenNumA.
takenLin.
schoolCalc.
takenM222.
```

$\overline{VII}$ Facts

$\Delta \equiv I - \overline{VII}$

Finally we put the question to the system:

>> mathReq.

which means:
whether a student completing $\overline{VII}$ satisfies maths requirement.

>> no. PROLOG Replies

So maths requirements **are not met**!

Weakness: we do not know why?

# LECTURE 4
## LOGIC PROGRAMMING

```
>> calcSeq.
   yes
>> discreteSeq.
   yes.
```

So we discover that at least one problem is that discrete sequence is not met.
In PL PROLOG there is some problem (SWI-implementation)
One can solve it by using predicate.

PROBLEMS (with above example)

- the answer is yes or no
  ( but why ? )
- this applies only to a particular student. One has to adjust it.

We need more advance Logic to permit to pass arguments to the propositional symbols:

PREDICATE LOGIC

# LECTURE 4
## LOGIC PROGRAMMING

COMMENTS about REFUTATION:

How to show unsatisfiability ?
of $\Delta \cup \{\sim Q\} \equiv \tilde{\Delta}$

**Th1:** $\Delta \models Q$ iff $\Delta \cup \{\sim Q\}$ unsatisfiable

If $\Delta$ is a **set of clauses** &

$\tilde{\Delta} = \Delta \cup \{\sim Q\}$ (i.e. $Q \equiv p_1 \wedge \dots \wedge p_n$)

then:

**Th2:** $\tilde{\Delta} = \Delta \cup \{\sim Q\}$ is unsatisfiable

iff

$\square \in \tilde{\Delta}^*$ (RESOLUTION CLOSURE)

$\square$ = False ← True = empty clause

PROOF: extended version to PREDICATE LOGIC shown later.

Thus

by Th.1 & Th.2 for $\tilde{\Delta}$ clause

$\Delta \models Q$ iff $\square \in \tilde{\Delta}^*$

derivable

$\tilde{\Delta} \vdash\!\!\rightarrow \square$ Resolution

HOW to find REFUTATION ? PROLOG has a special one.

# LECTURE 4

## LOGIC PROGRAMMING

Still how PROLOG reaches an empty clause $\square$ (a contradiction) is a mistery now.

But this version of REFUTATION is implemented in most PROLOG interpreters.

Note: this is theorem to be proved

$$\boxed{P \Rightarrow q} \equiv \sim(P \wedge \sim q) \equiv \sim q \Rightarrow \sim P$$

$\boxed{P \Rightarrow q}$: olirect proof

$\sim(P \wedge \sim q)$, $\sim q \Rightarrow \sim P$: indirect proofs.

FCI

a) **Forward chaining inference:**

has its analogone in direct maths theorem proving:

$$\left.\begin{array}{l} P \Rightarrow r_2 \\ r_1 = P \end{array}\right\} \text{ Modus Ponens } \quad r_2$$

$$\left.\begin{array}{l} r_2 \Rightarrow r_3 \\ r_2 \end{array}\right\} \text{ Modens Ponens } \quad r_3$$

# LECTURE 4
## LOGIC PROGRAMMING

$r_3 \Rightarrow r_4$ }
$r_3$ } Modus Ponens
$r_4$

⋮

$r_{n-1} \Rightarrow q$ }
$r_{n-1}$ } Modus Ponens
$q$

finally we infer $q$
in direct proof
(the same as FCI)

b) indirect proof in maths for $p \Rightarrow q$
($p$: assumption(*), $q$: claim)

$\equiv \sim(p \wedge \sim q)$
↑

this means assumptions & negation of claim of (*) yield contradiction.

The latter is equivalent to say

$\Delta \cup \{\sim Q\}$ is unsatisfiable.

REFUTATION is INDIRECT PROOF.

# LECTURE 4
# LOGIC PROGRAMMING

**Warning:**

$$\left.\begin{array}{l} p \leftarrow q, r \\ q, r \leftarrow \end{array}\right\} S$$

$$\frac{}{p} \quad ?!$$

Two literals in one step of refutation cannot be cancelled!

Otherwise as REFUTATION IS SOUND we would have:

$$S \models p.$$

so each model for S is a model for p.

But for $I(q) = \text{true}$, $I(p) = \text{false}$

$I(r) = \text{false}$

$$\left.\begin{array}{l} I(p \vee \sim q \vee \sim r) = \text{true} \\ I(q \vee r) = \text{true} \end{array}\right\} \Rightarrow \text{I is a model for S.}$$

But I is not a model for p.

A contradiction.

# LECTURE 4
## LOGIC PROGRAMMING

PROLOG uses special REFUTATION:

**SLD - REFUTATION**

We explain later this notion.

**Example 6:**

$$\Delta = \begin{cases} Q :- P. & (1) \\ P. & (2) \end{cases}$$

By **Modus Ponens**

$$\Delta \models Q.$$

Also can be shown by **REFUTATION:**

**Step 0:** $\tilde{\Delta} = \Delta \cup \{\sim Q\}$

$\Vert$

$\leftarrow Q$

**step 1:** $Q \leftarrow P$ (1)

$\leftarrow Q$

$\leftarrow P$ RESOLVENT 1

**Step 2:** $\leftarrow P$

$P \leftarrow$

$\square$ RESOLVENT 2 contradiction

Thus $\square \in \tilde{\Delta}^* \Leftrightarrow \Delta \models Q.$

$\Vert$

$\Delta \cup \{\sim Q\}$