People rarely learn unless they have fun in what they are doing

Dale Carnegie

About me

- Lecturer: Lela (Theodora)Koulouri
  - lela.koulouri@kcl.ac.uk
  - Office hours: **Friday**, **2-4pm**
    - Room: **S6.06**
  - https://lela.youcanbook.me/

- Email me for any personal/individual issue…

The Discussion Board on KEATS

- For any question, use the **Discussion Board** first.
- Give opportunity for peers to think about, and to provide a answer
- Everyone can read and benefit from the discussion
- We always respond within 48 hours

4CCS1ELA ELEMENTARY LOGIC WITH APPLICATIONS

LECTURE 8:
PROPOSITIONAL DEFINITE CLAUSE PROGRAMMING

Previously in ELA and what's next

- So far you have studied:
  - Propositional and first order logic
  - Truth tables and natural deduction

- In next three lectures:

Transform propositional and first order logic formulas into **programs**!

The ‘Programming with Logic’ Lectures

- The material in these three lectures is taken from a number of different sources.
- If you study the material in the lecture slides and tutorials, this will be sufficient for the exams!
- You will revisit and build on these lectures next year in 5CCS2PLD.
  - For those interested, I can also recommend more advanced reading material.

Objectives for the day

- Remember Conjunctive Normal Form
- Learn about and work on Propositional Horn and Definite Clause Rules
- Learn about and work on Programming with Definite Clause Rules

Feel free to ask questions any time!

Conjunctive Normal Form(CNF)

Conjunctive Normal Form (CNF)

- CNF is a **conjunction** of one or more formulas, each of which is a **disjunction** of one or more literals
  - Simply, CNF is an **AND** of **ORs**

(remember: a **literal** is a propositional **variable** or the negation of a propositional variable)

- The only connectives permitted are: $\wedge$ $\vee$ $\neg$
- **Note:** The $\neg$ operator can only be used as part of a literal.

$$(A \vee B) \wedge C$$

Example

- Which formula is in CNF and which one is not?

- $(P \vee \neg Q) \wedge (\neg P \vee Q)$
- $(P \rightarrow R) \wedge (\neg P \vee Q)$
- $\neg(P \vee Q) \wedge T$

Example

- Which formula is in CNF and which one is not?

- $(P \vee \neg Q) \wedge (\neg P \vee Q)$ is in CNF
- $(P \rightarrow R) \wedge (\neg P \vee Q)$ is **not** in CNF
- $\neg(P \vee Q) \wedge T$ is **not** in CNF

Conjunctive Normal Form (CNF)

- CNF is a **conjunction** of one or more formulas, each of which is a **disjunction** of one or more literals
  - Simply, CNF is an **AND** of **ORs**

$$(A \lor B) \land C$$

- A conjunction of literals is in CNF
  - Because it is like a conjunction of one-literal formulas
  - $A \land B$
    - *The example is a conjunction of two formulas with one literal each.*
- A disjunction of literals is in CNF
  - Because it is like a conjunction of a single formula
  - $A \lor B$
    - *The example is a conjunction of just one disjunction.*

Exercise

Which of the following formulae are in **CNF**, and why?

1. $\neg Q \lor (S \land P)$
2. $P \lor Q$
3. $(\neg Q \lor S) \land (\neg Q \lor P)$
4. $P \land Q$
5. P

Exercise

Which of the following formulae are in **CNF**, and why?

1. $\neg Q \vee (S \wedge P)$
2. $P \vee Q$
3. $(\neg Q \vee S) \wedge (\neg Q \vee P)$
4. $P \wedge Q$
5. P

Transform a formula in CNF

- Use equivalence rules
- Apply them until you get to CNF

1) $F \rightarrow G \equiv \neg F \vee G$
2) $F \leftrightarrow G \equiv (F \rightarrow G) \wedge (G \rightarrow F)$
3) $\neg(F \vee G) \equiv \neg F \wedge \neg G$
4) $\neg(F \wedge G) \equiv \neg F \vee \neg G$
5) $\neg\neg F \equiv F$
6) $F \vee (G \wedge H) \equiv (F \vee G) \wedge (F \vee H)$
7) $(F \wedge G) \vee H \equiv (F \vee H) \wedge (G \vee H)$

(F,G and H are propositional formulas and not literals)

Example

- Transform this formula into CNF:

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

We apply **Rule 6)** and we get:

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

Exercise

- Transform the following formulas into CNF:

1. P
2. $P \rightarrow Q \wedge R$
3. $Q \rightarrow S$
4. $\neg(S \wedge R) \vee T$

Exercise

- Transform the following formulas into CNF:

1. P *(already in CNF)*

1. $P \to Q \wedge R$
2. $Q \to S$
3. $\neg(S \wedge R) \vee T$

Exercise

- Transform the following formula into CNF:

1. $P \rightarrow Q \wedge R$

$\equiv \neg P \vee (Q \wedge R)$ *(we applied 1))*

$\equiv (\neg P \vee Q) \wedge (\neg P \vee R)$ *(we applied 6))*

Now it is in CNF!

Exercise

- Transform the following formula into CNF:

2. $Q \rightarrow S$

$\equiv \neg Q \vee S$ *(we applied 1))*

In CNF!

Exercise

- Transform the following formula into CNF:

3. $\neg(S \land R) \lor T$

$\equiv (\neg S \lor \neg R) \lor T$ *(we applied 4))*

In CNF!

Associativity

- Remember the law of associativity:
- $P \vee (Q \vee R) \equiv (P \vee Q) \vee R$

- So:

$$(\neg S \vee \neg R) \vee T \equiv \neg S \vee (\neg R \vee T)$$

- It doesn't matter where we put the brackets
- So we can simply drop the brackets

$$(\neg S \vee \neg R) \vee T \equiv \neg S \vee (\neg R \vee T) \equiv \boldsymbol{\neg S \vee \neg R \vee T}$$

Back to the exercise

- We started with these formulas:

1. P
2. $P \to Q \land R$
3. $Q \to S$
4. $\neg(S \land R) \lor T$

- We transformed them into CNF. This is our '**Knowledge Base**' now:

KB1

1. P
2. $(\neg P \lor Q) \land (\neg P \lor R)$
3. $\neg Q \lor S$
4. $\neg S \lor \neg R \lor T$

Propositional Horn and Definite Clauses

Clause

- A **clause** is a disjunction of one or more literals.

- So let's rephrase the CNF definition:
  - CNF is a **conjunction** of one or more formulas, each of which is a **disjunction** of one or more literals. Or:
- CNF is a **conjunction** of one or more **clauses**!

Let's count the clauses in our KB

CNF is a **conjunction** of one or more **clauses** (disjunctions of 1 or more literals)

KB1

1. P
2. $(\neg P \vee Q) \wedge (\neg P \vee R)$
3. $\neg Q \vee S$
4. $\neg S \vee \neg R \vee T$

- CNF 1., 3., and 4. have one clause
- CNF 2. has 2 clauses:
  $(\neg P \vee Q)$ and $(\neg P \vee R)$

Horn clauses

KB1

1. P — 1 clause
2. $(\neg P \vee Q) \wedge (\neg P \vee R)$ — 2 clauses
3. $\neg Q \vee S$ — 1 clause
4. $\neg S \vee \neg R \vee T$ — 1 clause

- How many positive literals do you see in each clause?
  - Just one positive literal!
- **A Horn clause is a clause with no more than one positive literal**

Horn clauses

- Any propositional formula can be transformed into CNF
- But not all CNF formulas are made up *only* of Horn clauses…

Horn clauses

Is this formula in CNF? If not, transform it!

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

Not in CNF

Horn clauses

Is this formula in CNF? If not, transform it!

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

$\equiv (\neg S \vee \neg R) \wedge (P \rightarrow (Q \vee R))$ (rule 4)

$\equiv (\neg S \vee \neg R) \wedge (\neg P \vee (Q \vee R))$ (rule 1)

$(\neg S \vee \neg R) \wedge (\neg P \vee Q \vee R)$ (rule 8) **In CNF!**

Horn clauses

- How many clauses does
$(\neg S \vee \neg R) \wedge (\neg P \vee Q \vee R)$ contain?
  - Two clauses
- Are they Horn clauses?
  - $(\neg P \vee Q \vee R)$ is not a Horn clause (count the positive literals – there are two!)

Definite clauses

KB1

1. P
2. $(\neg P \lor Q) \land (\neg P \lor R)$
3. $\neg Q \lor S$
4. $\neg S \lor \neg R \lor T$

- We saw that each Horn clause in the above CNF formulas has exactly one positive literal.

- **A definite clause is a Horn clause with exactly one positive literal.**

Definite clauses

- Back to the previous example:

$(\neg S \lor \neg R) \land (\neg P \lor Q \lor R)$

- *We said that* $(\neg P \lor Q \lor R)$ is **not** a Horn clause but $(\neg S \lor \neg R)$ is a Horn clause.
- Is $(\neg S \lor \neg R)$ also a definite clause?
- No! it has 0 positive literals!

Exercise

In the following CNF formula which clauses are **Horn clauses** and which are **definite clauses**?

$$(\neg Q \lor \neg S) \land (\neg Q \lor P) \land (\neg Q \lor P \lor R)$$

Exercise

In the following CNF formula which clauses are **Horn clauses** and which are **definite clauses**?

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

HORN

DEFINITE

NEITHER

Definite clauses: formal procedure

1.Take any propositional well formed formula (wff) α
(from now on greek letters α (alpha), β (beta), γ (gamma) will denote any wff i.e. α stands for $\neg P$, $P \wedge Q$, $P \rightarrow Q$ …)

2.Transform α to CNF. Let's write '*cnf*(α)' to denote the transformation of α to CNF

3.*cnf*(α) is a formula of the form:
$\boldsymbol{\beta_1 \wedge \ldots \wedge \beta_n}$
where $n \geq 1$, and each $\boldsymbol{\beta}_i$ is a clause
(a disjunction of literals: positive or negative propositional variables)

4. If $\boldsymbol{\beta}_i$ is of the form
$\neg \mathbf{X}_1 \vee \ldots \vee \neg \mathbf{X}_m \vee \mathbf{X}$, $\boldsymbol{m \geq 0}$,
(exactly one positive and 0 or more negative literals) then $\boldsymbol{\beta}_i$ is **definite clause**

From definite clauses to **DEFINITE RULES**

- A definite clause of the form

$\neg \mathbf{X}_1 \vee \ldots \vee \neg \mathbf{X}_m \vee \mathbf{X}$

can be represented as the equivalent:

$\mathbf{X}_1 \wedge \ldots \wedge \mathbf{X}_m \rightarrow \mathbf{X}$

- Can you see why ?
- $\neg(F \wedge G) \equiv \neg F \vee \neg G$
- $F \rightarrow G \equiv \neg F \vee G$

From definite clauses to **DEFINITE RULES**

$\neg X_1 \vee \ldots \vee \neg X_m \vee X$

$\equiv (\neg X_1 \vee \ldots \vee \neg X_m) \vee X$

$\equiv \neg(X_1 \wedge \ldots \wedge X_m) \vee X$

$\equiv \mathbf{X_1 \wedge \ldots \wedge X_m \rightarrow X}$

Example

KB1

1. P
2. $(\neg P \vee Q) \wedge (\neg P \vee R)$
3. $\neg Q \vee S$
4. $\neg S \vee \neg R \vee T$

- First, list all **definite clauses** of these CNF formulas
- Then, represent the definite clauses as **rules**

Definite clauses		Rules
P	$\rightarrow$	$\rightarrow P$
$(\neg P \vee Q)$		$P \rightarrow Q$
$(\neg P \vee R)$		$P \rightarrow R$
$\neg Q \vee S$		$Q \rightarrow S$
$\neg S \vee \neg R \vee T$		$(S \wedge R) \rightarrow T$

Formulas-CNF-Definite Clauses-Definite Rules... Why all this hassle?

- **We program with definite rules!**

- Definite clause programming is the basis of *logic programming*

Logic programming

Logic Programming

- In a **procedural** style of programming, the program explicitly describes the **individual steps** of computation.
  - Examples: Imperative programming (C), object-oriented programming (Java)
- In contrast, **Logic programming** is a **declarative** style of programming.
  - The programmer says **what** they want to compute, but does not explicitly specify **how** to compute it.
    - It is up to the interpreter (compiler/implementation) to figure out how to perform the computation requested.
  - Examples: Logic programming (Prolog), database query languages (SQL), functional programming (Haskell)

Logic programming

- A logic program is given as a set of assumed properties (stated as logical **formulas**) about the world (or rather about the world of the program)
  - What we just called a knowledge base!
- The user supplies a logical formula stating a property that might or might not hold in the world as a **query**
- The system determines whether the queried property is a **consequence** of the assumed properties in the program.

Consequence

- We say that a formula G is a **logical consequence** of formulas $F_1, \ldots, F_n$ (or, G logically follows from $F_1, \ldots, F_n$), if in every interpretation where each and every of $F_1, \ldots, F_n$ evaluates to 1, G also evaluates to 1.

- Notation:

$F_1, \ldots, F_n \models G$

- Examples:

$p, \; q \lor \neg p, \; \neg p \lor \neg q \models q$

Example

1. If I am in the office, then you can pop in if I am not too busy.
2. If I am in the office, then I am not too busy.
3. *Therefore, if I am in the office, then you can pop in.*

Let

L stand for ‘I am in the office’,

P stand for ‘you can pop in’

B stand for ‘I am too busy’

Write the premises and conclusion using formal language

Example

1. If I am in the office, then you can pop in if I am not too busy.
2. If I am in the office, then I am not too busy.
3. Therefore, if I am in the office, then you can pop in.

Let

L stand for 'I am in the office',
P stand for 'you can pop in'
B stand for 'I am too busy'

$$L \rightarrow (\neg B \rightarrow P)$$

$$L \rightarrow \neg B$$

$$\therefore L \rightarrow P$$

Does the *conclusion* logically follows (is a consequence of) the *premises*?

Deciding consequence using a Truth-table

	L	P	B	¬B	¬B → P	L → (¬B → P)	L → ¬B	L → P
1	1	1	1	0	1	1	0	1
2	1	1	0	1	1	1	1	1
3	1	0	1	0	1	1	0	0
4	1	0	0	1	0	0	1	0
5	0	1	1	0	1	1	1	1
6	0	1	0	1	0	1	1	1
7	0	0	1	0	1	1	1	1
8	0	0	0	1	0	1	1	1

Deciding consequence using a Truth-table

	L	P	B	$\neg B$	$\neg B \rightarrow P$	$L \rightarrow (\neg B \rightarrow P)$	$L \rightarrow \neg B$	$L \rightarrow P$	
1	1	1	1	0	1	1	0	1	
2	1	1	0	1	1	1	1	1	☆
3	1	0	1	0	1	1	0	0	
4	1	0	0	1	0	0	1	0	
5	0	1	1	0	1	1	1	1	☆
6	0	1	0	1	0	1	1	1	☆
7	0	0	1	0	1	1	1	1	☆
8	0	0	0	1	0	1	1	1	☆

In every interpretation in which both premises are true, the conclusion is also true, so: $L \rightarrow P$ is a logical consequence of $L \rightarrow (\neg B \rightarrow P)$ and $L \rightarrow \neg B$)

Deciding consequence

- You can use truth-tables and natural deduction to decide whether G is a logical consequence of $F_1, \ldots, F_n$
- But truth-tables can get very large very quickly
  - A truth table with 10 variables has 1024 rows
  - A truth table with 80 variables would take 38 billion years to make (if 1 millionth of a second is needed to make one row)
- Natural deduction is unsuitable for automation - it requires ingenuity in deciding which rule to apply.
- So we need an algorithm – an efficient one!

Definite Clause Propositional Programming

Going back to our example

1. P
2. $(\neg P \vee Q) \wedge (\neg P \vee R)$
3. $\neg Q \vee S$
4. $\neg S \vee \neg R \vee T$

KB1

- First, list all **definite clauses** of these CNF formulas
- Then, represent the definite clauses as **rules**

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

$\longrightarrow$

$\rightarrow P$
$P \rightarrow Q$
$P \rightarrow R$
$Q \rightarrow S$
$(S \wedge R) \rightarrow T$

Propositional definite clause programming

$\boxed{\mathscr{P}1}$

$$\rightarrow P$$
$$P \rightarrow Q$$
$$P \rightarrow R$$
$$Q \rightarrow S$$
$$S \wedge R \rightarrow T$$

- Is T a logical consequence of $\mathscr{P}1$?

$\mathscr{P}1 \models T$?

Propositional Definite clause programming

$\mathcal{P}1$

$$\rightarrow P$$
$$P \rightarrow Q$$
$$P \rightarrow R$$
$$Q \rightarrow S$$
$$S \wedge R \rightarrow T$$

Body

Head

$$?T$$

Query

Propositional Definite clause programming

$\mathscr{P}1$

$$\to P$$

$$P \to Q$$

$$P \to R$$

$$Q \to S$$

$$S \wedge R \to T$$

Body

Head

$? \; T$

Query

- Look for a rule with head *T* and replace query ? *T* with body of rule (expand *T*)

- Select *any* literal in the new query and repeat

Propositional Definite clause programming

$\mathcal{P}1$

$\to P$

$P \to Q$

$P \to R$

$Q \to S$

$S \wedge R \to T$

$?\ T$

$?\ S \wedge R$

$?\ Q \wedge R$

$?\ P \wedge R$

$?\ R$

$?\ P$

$?$

Success

Propositional Definite clause programming

$\mathscr{P}1$

$$\rightarrow P$$
$$P \rightarrow Q$$
$$P \rightarrow R$$
$$Q \rightarrow S$$
$$S, R \rightarrow T$$

Replace the ∧ with ,

$? \; T$

$? \; S, R$

$? \; Q, R$

$? \; P, R$

$? \; R$

$? \; P$

$?$

Success

Propositional Definite clause programming

$\mathcal{P}1$

$\rightarrow P$

$P \rightarrow Q$

$P \rightarrow R$

$Q \rightarrow S$

$S, R \rightarrow T$

Although choice doesn't matter, **you must be consistent in query literal selection strategy** (e.g. always **leftmost** literal, or always rightmost)

$?\ T$

$?\ S, R$

$?\ Q, R$

$?\ P, R$

$?\ R$

$?\ P$

$?$

Success

Propositional Definite clause programming

$\mathcal{P}1$

$\rightarrow P$
$P \rightarrow Q$
$P \rightarrow R$
$Q \rightarrow S$
$S, R \rightarrow T$

This choice **can** determine whether proof succeeds or fails (there may be more than one such rule)

$? \; T$
$? \; S, R$
$? \; Q, R$
$? \; P, R$
$? \; R$
$? \; P$
$?$
Success

Derivation tree

$\mathcal{P1}$

$$\rightarrow P$$
$$P \rightarrow Q$$
$$P \rightarrow R$$
$$Q \rightarrow S$$
$$S , R \rightarrow T$$

Query $\mathcal{P1}$

$?\, T$	$S , R \rightarrow T$
$?\, S , R$	$Q \rightarrow S$
$?\, Q , R$	$P \rightarrow Q$
$?\, P , R$	$\rightarrow P$
$?\, R$	$P \rightarrow R$
$?\, P$	$\rightarrow P$
$\square$	

SUCCEEDS

leftmost query selection always applied

Definite clause programming: formal procedure

- Let $\boldsymbol{P}$ be a definite clause program containing definite rules of the form $X_1, \dots, X_n \rightarrow Y$
- Let $?\, Q_1, \dots, Q_m$ be a query on P

1. Choose a literal $\boldsymbol{Q_i}$

2. *If* there is **no rule** $X_1, \dots, X_n \rightarrow Q_i$ in *P* then **exit** and **fail**,
   else **choose** a rule $X_1, \dots, X_n \rightarrow Q_i$ in P

3. In the query term
   $\mathbf{?}\, Q_1, \dots, Q_{i-1}, Q_i, Q_{i+1}, \dots, Q_m$
   replace Q_i by $X_1, \dots, X_n$:
   $\mathbf{?}\, Q_1, \dots, Q_{i-1}, X_1, \dots, X_n, Q_{i+1}, \dots, Q_m$

4. *If* the query term is empty then **exit** and **success**,
   else go to step 1 and repeat 1 - 4.

Exercise

- P
- $P \vee Q \rightarrow S$

1. Transform the formulas to CNF
2. then to definite clause program
3. Then draw two derivation trees for the query $?S$ one that fails and one that succeeds.

1. Transform to CNF

- P

Already in CNF

- $P \vee Q \rightarrow S$

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

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

$\boldsymbol{(\neg P \vee S) \wedge (\neg Q \vee S)}$

Now in CNF

2. Transform to definite clause program

KB1

P

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

- First, list all **definite clauses** of these CNF formulas
- Then, represent the definite clauses as **rules**

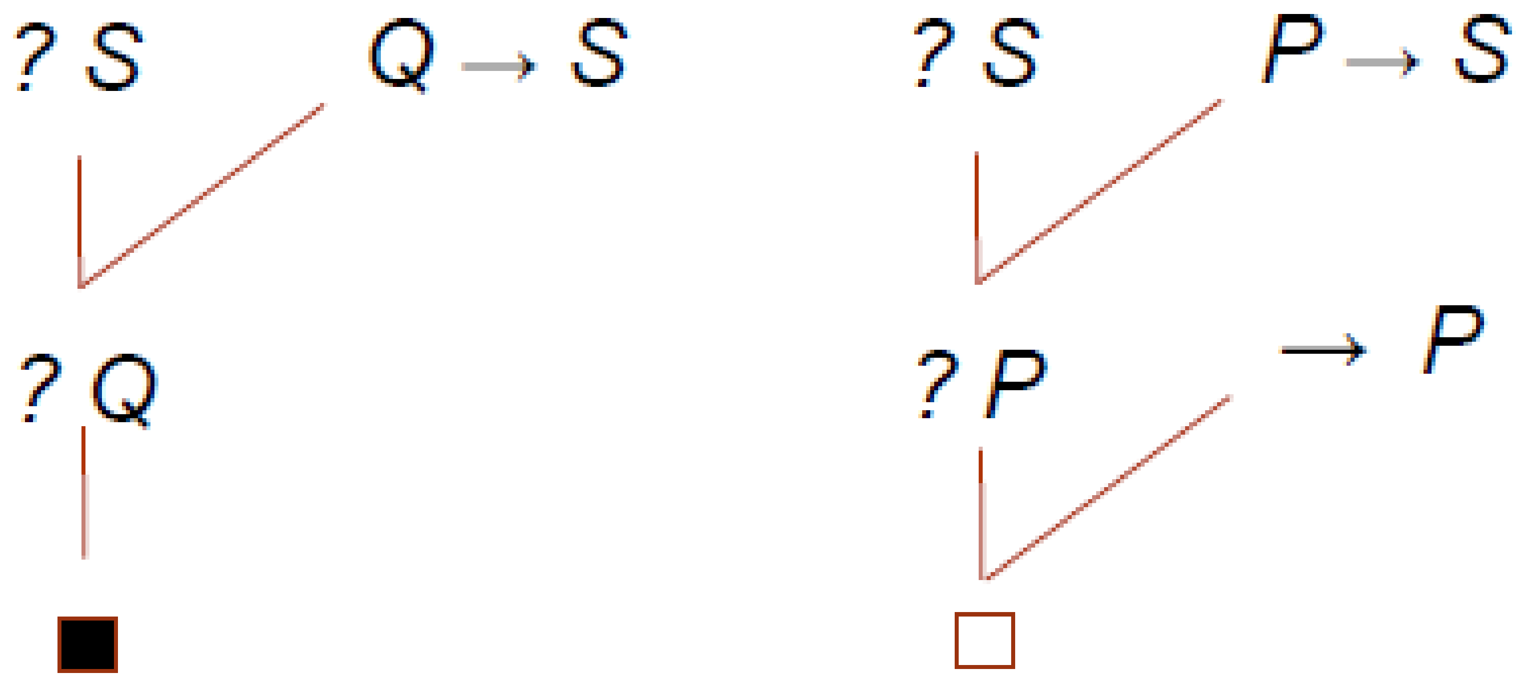
P
$(\neg P \vee S)$
$(\neg Q \vee S)$

$\longrightarrow$

$\mathscr{P}1$

$\rightarrow P$
$P \rightarrow S$
$Q \rightarrow S$

3. draw derivation trees for ? S



FAILS SUCCEEDS

Tutorials and Next Lecture

- **Large Group Tutorial:**
  - Repeats *lecture exercises* in today's slides – make sure you can do them yourself !
  - *Tutorial questions 3 and 4* not in slides
- **Small Group Tutorials:**
  - This week: Work on Questions 1 and 2 only

- **Next Lecture:**
  - Prenex Normal Form
  - First order definite clause programs