UNIT-IV (Part 1)

Logic Programming

Logic Programming

1. Relations
2. First Order Logic
3. Logic Programming and Horn-Clause Programming
4. Unification
5. Deduction and Search as a strategy for deduction,
6. Indexing, Pruning,
7. Definite Clause Grammars.
8. Case Study on Logic Programming with Prolog

Relations

Topics: View of Relation

Sample relation “append”

Relations – specified by Rules and Facts (special Rules)

Queries on Relation

(Ref : Programming Languages by Ravi Sethi Pg 426)

View of RELATION

- **Relation** can be viewed as a TABLE
  - with n>=0 columns, and possibly infinite rows
- A tuple (a1,a2,..,an) is in relation, if ai appears in ith column, 1<=i<=n, for some row in the relation Table

Sample Relation : “append”

Relation append is a set of tuples of the form (X,Y,Z),where Z contains elements of X, followed by elements of Y. Few tuples in relation **append** are as follows:

X	Y	Z
[]	[]	[]
[a]	[]	[a]
…	….	….
[a,b]	[c,d]	[a,b,c,d]
…	….	….

RULES and FACTS

Relations are Specified by **RULES** written in pseudocode as :

P if Q1 and Q2 and….and Qk.   Where k>=0

Such RULES are called HORN CLAUSES

(More on Horn Clauses later)

FACTS : Special case of Rules with k = 0

i.e. P holds without any condition, so written simply as P

Horn Clause

- Named after person who studied them (Alfred Horn)
- Horn Clauses lead to Efficient Implementation

Sample Relation **append** - Specified by 2 Rules

2 Rules for append **Relation** (**PROLOG**)

```
append([],Y,Y)
append([H|X],Y,[H|Z]:-append(X,Y,Z)
```

Queries

- Logic Programming is Driven by Queries about **Relations**
- Simple query: Does a tuple belong to a relation?
  - e.g. Does ([a,b], [c,d], [a,b,c,d]) belong to **Relation** append?
  - Answer: yes

Queries with **Variables**

Q) Is there a Z such that append [a,b] and [c,d] get a Z?

Answer: yes, when Z=[a,b,c,d]

- The above is actually a request for answer to Z.
- Also, a query for X can be:

Q) Is there an X such that append X and [c,d] get [a,b,c,d]?

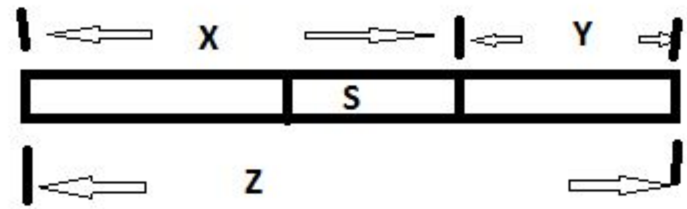
Contd..Queries with Variables

- Also, a query for Y can be:

Q) Is there an Y such that append [a,b] and Y get [a,b,c,d]?

- **Thus, there are several different ways to use a Relation** (like for append)

New Relations can be defined from Old



- E.g. S,X,Y,Z refer to portions of list
- New relations **prefix**, **suffix**, and **sublist**

prefix X of Z

  if for some Y, **append** X and Y to get Z.

suffix Y of Z

  if for some X, **append** X and Y to get Z.

sublist S of Z

  if for some X, **prefix** X of Z **and** **suffix** S of X.

First Order Logic (FOL)

Also called **Predicate Logic**

Topics: FOL vs Propositional Logic

Proposition

- A proposition is a logical statement that is only made if it is true
  - Today is Tuesday
  - The Earth is round

- Two forms of propositions are
  - Atomic propositions
  - Compound terms (multiple propositions connected through the logical operators of and, or, not, and implies)

Proposition

- A proposition is a logical statement that is only made if it is true (usefulness)
  - Today is Tuesday
  - The Earth is round

- Two forms of propositions are
  - **Atomic** propositions
  - **Compound** terms (multiple propositions connected through the **logical operators of and, or, not, and implies**)

Propositional Logic

- Not powerful enough to represent all types of assertions that are used in computer science and mathematics
- Not capable of expressing certain types of relationship between propositions such as equivalence.

  Example, **the assertion "x is greater than 1", where x is a variable, is not a proposition because you can not tell whether it is true or false unless you know the value of x.**

- **Thus the propositional logic can not deal with such sentences.**

- **However, such assertions appear quite often in mathematics and we want to do inferencing on those assertions.**

….Propositional Logic

Also **the pattern involved in the following logical equivalences can not be captured by the propositional logic:**

"Not all birds fly" is equivalent to "Some birds don't fly".

"Not all integers are even" is equivalent to "Some integers are not even“

"Not all cars are expensive" is equivalent to "Some cars are not expensive"

NEED more Powerful Logic to deal with these and other problems

Limitations (1) of Propositional Logic

Statements that hold for **many objects** must be enumerated

- John is a CS UC graduate -> John has passed cs441
- Ann is a CS UC graduate ->Ann has passed cs441
- Ken is a CS UC graduate -> Ken has passed cs441

Solution : use **Variables**

- **x** is a CS UC graduate -> **x** has passed cs441

Limitations (2) of Propositional Logic

Statements that define the property of the **group of objects.**

Example:

All new cars must be registered

Some of the CS graduates graduate with distinction

Solution: Make statements with **quantifiers**

- **Universal quantifier** –the property is satisfied by all members of the group
- **Existential quantifier** – at least one member of the group satisfy the property

Predicate Logic (First Order Logic)

- Explicitly models **objects** and their **properties**

- Allows to make **statements with variables** and **quantify** them

Predicate

- Predicates represent properties or relations among objects
- A predicate P(x) assigns a value true or false to each x depending on whether the property holds or not for x.
- The assignment is best viewed as a **big table** with the variable x substituted for objects from the **universe of discourse**

Predicate Logic

- **Constant** – Models a specific object
  - **Examples**: “John”, “France”, “7”
- **Variable** – represents object of specific type (defined by the universe of discourse)
  - **Examples**: x, y (universe of discourse can be people, students, numbers)
- **Predicate** - over one, two or many variables or constants.
  - Represents properties or relations among objects
    Examples: Red(car23), student(x), married(John,Ann)

Predicate : Example

Example:

Assume Student(x) where the universe of discourse are **people**

- Student(John) …. ?T/F (if John is a student)
- Student(Ann) …. ?T/F (if Ann is a student)
- Student(Jane) …. ?T/F (if Jane is not a student)

Predicate : Example

- Predicate P(x) : **x is a prime number**
- Truth values for different x:
  - P(2) T
  - P(3) T
  - P(4) F
  - P(5) T
  - P(6) F

- P(2), P(3), P(4), P(5), P(6) are propositions?
- **P(x) with variable x** is **not a proposition?**

Sample Statements

Bob is Fred’s father	father(Bob, Fred)
Sue is Fred’s mother	mother(Sue, Fred)
Barbara is Fred’s sister	sister(Barbara, Fred)
Jerry is Bob’s father	father(Jerry, Bob)

Sample Statements (and)

A person's father's father is the person's grandfather

First-order predicate calculus as:

x, y, z : if father(x, y) and father(y, z) then grandfather(x, z)

Rewrite these as

grandfather(x, z) $\subset$ father(x, y) and father(y, z)

Sample Statements (or)

A person's father or mother is that person's parent

x, y : if father(x, y) or mother(x, y) then parent(x, y)

$$\text{parent}(x, y) \subset \text{father}(x, y)$$

$$\text{parent}(x, y) \subset \text{mother}(x, y)$$

Quantified statements

- Predicate logic lets us make statements about **groups of objects**
  - To do this we use special quantified expressions
- Two types of quantified statements:
  - • **Universal**
  - Example: ‘ all CS VIIT graduates have to pass cs441” – the statement is true for all graduates
  - • **Existential**
  - Example: ‘Some CS VIIT students graduate with honor.’ – the statement is true for some people

Predicates vs Propositions

- Predicate P(x,y,z) vs Propositions (T/F)
  - x,y,z are variables so P(x,y,z) as such, cannot be associated with either T/F value
- Predicates can be converted to Propositions by INSTANTIATIONS of variables

  and by using universal/existential quantifiers ($\forall$ / $\exists$)
  - Example : $\forall \mathbf{x}\ \exists y\ \exists z\ P(x,y,z) \equiv$ For all x, there exists a 'y' , there exist a 'z' for which P is TRUE

Universal quantifier

Quantification converts a propositional function into **a proposition** by binding a variable to a set of values from the universe of discourse.

Example:

- Let P(x) denote x > x - 1. Assume x are real numbers.
- Is P(x) a proposition? **No.** Many possible substitutions.
- Is $\forall$x P(x) a proposition? **Yes.**
- What is the truth value for $\forall$x P(x) ?
  - **True**, since P(x) holds for all x.

Existential quantifier

Quantification converts a propositional function into **a proposition** by binding a variable to a set of values from the universe of discourse.

Example:

- Let T(x) denote x > 5 and x is from Real numbers.
- Is T(x) a proposition? **No.**
- Is $\exists$x T(x) a proposition? **Yes.**
- What is the truth value for $\exists$x T(x) ?
  - Since 10 > 5 is true. Therefore, $\exists$x T(x) is **true.**

Summary of quantified statements

- When $\forall$**x P(x)** and $\exists$**x P(x)** are true and false?

Statement	When true?	When false?
$\forall x\, P(x)$	P(x) true for all x	There is an x where P(x) is false.
$\exists x\, P(x)$	There is some x for which P(x) is true.	P(x) is false for all x.

Suppose the elements in the universe of discourse can be enumerated as x1, x2, ..., xN then:

- **$\forall$x P(x) is true whenever P(x1) $\wedge$ P(x2) $\wedge$ ... $\wedge$ P(xN) is true**
- **$\exists$x P(x) is true whenever P(x1) $\vee$ P(x2) $\vee$ ... $\vee$ P(xN) is true**.

Points to remember:

- The main connective for universal quantifier $\forall$ is implication $\rightarrow$.
- The main connective for existential quantifier $\exists$ is and $\wedge$.

Translation with quantifiers

Sentence:

- Someone at VIIT is smart.

Assume: Domain is VIIT affiliates:

Translation:

- $\exists$ x Smart(x)

Assume: Domain is people:

- $\exists$ x at(x,VIIT) $\wedge$ Smart(x)

Translation with quantifiers

- **Assume two predicates S(x) and P(x)**

Universal statements typically tie with implications

- **All S(x) is P(x)**
  - $\forall x\ (S(x) \rightarrow P(x))$
- **No S(x) is P(x)**
  - $\forall x(S(x) \rightarrow \neg P(x))$

Existential statements typically tie with conjunctions

- **Some S(x) is P(x)**
  - $\exists x\ (S(x) \wedge P(x))$
- **Some S(x) is not P(x)**
  - $\exists x\ (S(x) \wedge \neg P(x))$

Nested quantifiers

- More than one quantifier may be necessary to capture the meaning of a statement in the predicate logic.

Example:

- Every real number has its corresponding negative.
- **Translation:**
  - Assume:
    - a real number is denoted as x and its negative as y
    - A predicate P(x,y) denotes: "$x + y = 0$"
- Then we can write:

  $\forall x \, \exists y \, P(x,y)$

Properties of Quantifiers:

- In universal quantifier, $\forall x \forall y$ is similar to $\forall y \forall x$.
- In Existential quantifier, $\exists x \exists y$ is similar to $\exists y \exists x$.
- $\exists x \forall y$ is not similar to $\forall y \exists x$.

How to use these Predicates to Model Actual Scenarios

- Learn the syntax of the Logic

Syntax: FOL

- Constants
  - A|5|Pune|………
- Variable
  - a|s|x|…………..
- Predicate
  - Before | HasColor |Parent|……….
- Function
  - Mother | Cosine | HeadofList

Predicates vs Functions

- Predicates : When arguments of predicated are instantiated – predicate can have TRUTH value T/F

- Functions : When arguments of functions are instantiated function can have Any Return Value
  - numeric/non-numeric/ character strings / etc.

Sample (Everyone loves their Mother): Predicate vs Function

- Predicate - Mother

Mother(x,y) ≡ y is Mother of x

$\forall x\ \exists y\ Mother(x,y)\ \wedge\ Loves(x,y)$

- Function – Mother

$\forall x\ Loves(x, Mother(x))$

Note: Functions appear Only as ARGUMENTS to Predicates

…contd…Syntax: FOL

- Sentence **->** AtomicSentence

  |Sentence Connective Sentence

  |Quantifier Variable, …Sentence

  | ¬ Sentence | (Sentence)

AtomicSentence **->** Predicate(Term,..)

|Term=Term

Term **->** Function(Term,..) |Constant |Variable

Connective **->** $\vee$ | $\wedge$ | $\Rightarrow$ |<=>

Quantifier **->** $\forall$ **| ∃**

Examples of FOL using quantifier

- **All birds fly.**
  Predicate is "**fly(bird)**."
  And since there are all birds who fly so it will be represented as follows.
  $$\forall x \; \mathbf{bird(x) \rightarrow fly(x)}$$
- **Every man respects his parent.**
  Predicate is "**respect(x, y)," where x=man, and y= parent**.
  Since there is every man so will use $\forall$, and it will be represented as follows:
  $$\forall x \; \mathbf{man(x) \rightarrow respects\ (x, parent)}$$

…contd… Examples of FOL using quantifier

- **Some boys play cricket.**
  Predicate is "**play(x, y)**," where x= boys, and y= game.

- Since there are some boys so we will use ∃**, and it will be represented as**:

$$\exists x \; \mathbf{boys(x) \rightarrow play(x, cricket)}$$

…contd… Examples of FOL using quantifier

- **Not all students like both Mathematics and Science.**
  Predicate is "**like(x, y)," where x= student, and y= subject**.
  Since there are not all students, so we will use ∀ **with negation, so** following representation for this:
  ¬∀ (x) [student(x) → like(x, Mathematics) ∧ like(x, Science)]

Formulate into FOL

1) **Not all students take both History & Biology**

Student(x) ≡ x is a Student

Takes(x,y) ≡ x subject is taken by student y

$$\neg[\forall x\ \mathbf{Student(x)} \Rightarrow \mathbf{Takes(History,x)} \wedge \mathbf{Takes(Biology,x)}]$$

Equivalent formulation

$$\exists x\ \mathbf{Student(x)}\ \wedge\ [\neg \mathbf{Takes(History,x)}\ \vee\ \neg \mathbf{Takes(Biology,x)}]$$

Formulate into FOL

2) Only 1 student failed in History

Failed(x,y) ≡ Student y failed in subject x

$$\exists x\,[Student(x) \land Failed(History,x) \land \forall y\,[(\neg (x=y) \land Student(y)) \Rightarrow \neg Failed(History,y)]]$$

Formulate in FOL

3) Only one student failed in Both History and Biology

$\exists x\ [Student(x) \wedge Failed(History,x) \wedge Failed(Biology,x) \wedge \forall y\ [(\neg (x=y) \wedge Student(y)) \Rightarrow \neg Failed(History,y) \vee \neg Failed(Biology,y)\ ]]$

$\exists x\ [Student(x) \wedge Failed(History,x) \wedge Failed(Biology,x) \wedge \forall y\ [(\neg (x=y) \wedge Student(y)) \Rightarrow \neg (Failed(History,y) \wedge Failed(Biology,y))\ ]]$

Formulate in FOL

4) The best score in History is better than the best score in Biology

Function: score(Subject,Student)

Predicate : Greater(X,Y) ≡ X is greater than Y

- **$\forall$ x [Student(x) $\wedge$ Takes(Biology,x) $\Rightarrow$ $\exists$y [Student(y) $\wedge$ Takes(History,y) $\wedge$ Greater(score(History,y),score(Biology, x))]]**

Q) Translate using predicates and quantifiers

- “Every student in class has studied calculus”
- Rewrite: For every student in class, he has studied calculus
- With variable: For every student x in class, x has studied calculus
- Let, $C(x)$: “x has studied calculus”
- TRANSLATION= ?

Sample Conversions to FOL

All students are smart.	$\forall x \, (Student(x) \Rightarrow Smart(x))$
There exists a student.	$\exists x \, Student(x)$.
There exists a smart student.	$\exists x \, (Student(x) \wedge Smart(x))$
Bill is a student.	$Student(Bill)$
Bill takes either Analysis or Geometry (but not both).	$Takes(Bill, Analysis) \Leftrightarrow \neg Takes(Bill, Geometry)$
Bill takes Analysis or Geometry (or both).	$Takes(Bill, Analysis) \vee Takes(Bill, Geometry)$
Bill takes Analysis and Geometry.	$Takes(Bill, Analysis) \wedge Takes(Bill, Geometry)$
Bill does not take Analysis.	$\neg Takes(Bill, Analysis)$.
Bill has at least one sister.	$\exists x \, SisterOf(x,Bill)$
Bill has no sister.	$\neg \exists x \, SisterOf(x,Bill)$
Bill has at most one sister.	$\forall x, y \, (SisterOf(x, Bill) \wedge SisterOf(y, Bill) \Rightarrow x = y)$
Bill has exactly one sister.	$\exists x \, (SisterOf(x, Bill) \wedge \forall y \, (SisterOf(y, Bill) \Rightarrow x = y))$
Bill has at least two sisters.	$\exists x, y \, (SisterOf(x, Bill) \wedge SisterOf(y, Bill) \wedge \neg (x = y))$
Every student takes at least one course.	$\forall x \, (Student(x) \Rightarrow \exists y \, (Course(y) \wedge Takes(x,y)))$

Logic Programming and Horn-Clause Programming

Different paradigm for programming

- Declarative programming
  - Specify knowledge and how that knowledge is to be applied through a series of rules
  - Programming language environment uses one or more built-in methods to **reason over the knowledge** and **prove things** (**or answer questions**)
    - in logic programming, the common approach is to apply the methods of resolution and unification

Logic Programming

- Also known as Declarative programming

- A declarative program does not have code, instead it **defines two pieces of knowledge**
  - FACTS – statements that are true
  - RULES – if-then statements that are truth preserving

- Mostly synonymous with the **Prolog** language because it is the **only widely used language for logic programming**

…Logic Programming

- We **use the program to prove** if a **Statement is true** and/or **Answer questions**

- The reason this works is that Prolog has **built-in problem solving processes** called **Resolution** and **Unification**
  - Prolog is more of a Tool, but it does have *some* programming language features that make it mildly programmable

Logic Programming Languages

- While these languages have numerous flaws, they can build powerful problem solving systems with little programming expertise
  - Used extensively in AI research

Terminology

- Logic Programming is a specific type of a more general class: production systems (also called rule-based systems)
  - **Production system** is a collection of facts (knowledge), rules (which are another form of knowledge) and control strategies
  - Collection of **facts** (what we know) is referred to as working memory
  - **Rules** are simple if-then statements where the condition tests values stored in working memory and the action (then clause) manipulates working memory (adds new facts, deletes old facts, modifies facts)
  - **Control strategies** help select among a set of rules that match – If multiple rules have matching conditions, the control strategies can help decide which rule we select this time through
  - **Other control strategies** – whether we work from conditions to conclusions or from conclusions to conditions (forward, backward chaining respectively)

Logic Programming – Use of Program

- We use the **program** to prove if a Statement is true and/or Answer questions

Why Declarative Paradigm Works

- The reason this works is that :
  - Sample language Prolog has built-in problem solving processes called Resolution and Unification
  - Prolog is more of a Tool, but it does have *some* programming language features that make it mildly programmable

Background for Logic

- A proposition is a logical statement that is only made if it is true
  - Today is Tuesday
  - The Earth is round
- *Symbolic logic* uses propositions to express ideas, relationships between ideas and to generate new ideas based on the given propositions
- Two forms of propositions are
  - Atomic propositions
  - Compound terms (multiple propositions connected through the logical operators of and, or, not, and implies)
- Propositions will either be true (if stated) or something to prove or disprove (determine if it is true) – we do not include statements which are false
- For Symbolic logic, we use First order predicate calculus
  - Statements include predicates like round(x) where this is true if we can find an x that makes it true such as round(Earth) or round(x) when x = Earth
  - Predicate is like a Boolean function except that rather than returning T or F, it finds an x that makes it true

Logic Operators

Name	Symbol	Example	Meaning
negation	$\neg$	$\neg$ a	not a
conjunction	$\cap$	a $\cap$ b	a and b
disjunction	$\cup$	a $\cup$ b	a or b
equivalence	$\equiv$	a $\equiv$ b	a is equivalent to b
implication	$\supset$ $\subset$	a $\supset$ b a $\subset$ b	a implies b b implies a
universal	$\forall$ X.P		For all X, P is true
existential	$\exists$ X.P		There exists a value of X such that P is true

Meaning of the Logic Operators

- Equivalence means that both expressions have identical truth tables
- Implication is like an if-then statement
  - if a is true then b is true
  - Note: This does not necessarily mean that if a is false that b must also be false
- Universal quantifier ($\forall$) says that this is true no matter what x is
- Existential quantifier ($\exists$) says that there is an X that fulfills the statement

Horn Clause Topics

Disjunction and Conjunction
First Order formula in Clause Form
Restricted form of Clauses : HORN CLAUSE
Horn Clause and Prolog

Disjunction and Conjunction

$P1 \vee P2$.... (Disjunctions)

$Q1 \wedge Q2$...(Conjunctions)

Set of Statements in Clause Form

$\forall x1\ \forall x2..\ \forall xk\ (C1 \wedge C2 \wedge \ldots\ldots Cn)$

- **CONJUNCTION** of Clauses

Where, Ci is a clause made of **DISJUNCTIONS** of literals

e.g. $Ci = a1 \vee a2 \vee a3 \vee \neg a4 \vee \neg a5 \vee a6 \vee a7 \ldots.$

Each literal is an **atomic formula** or its **Negation**

CLAUSE FORM contains only Universal Quantifiers $\forall$, and that too bunched up in the left. Why?

Benefit of Clause Form

- Universal Quantifiers, bunched up in the left has the benefit that – They can be IGNORED (**Implicit**) during Processing
  - This makes program writing a **little bit** simpler

(More simple with further - Horn Clause)

Converting First Order Formula to Clause form

- Every First Order formula can be converted into CLAUSE FORM
  - Shown by Thoralf Skolem

Horn Clause

- Restricted form of CLAUSE
- A Horn clause is a clause with **at most one positive literal**.

Ex: $\neg D1 \vee \neg D2 \vee \neg D3 \vee \ldots\ldots \vee \neg Dk \vee D_{k+1}$

For convenience : Horn Clause can **also be seen as**

$(D1 \wedge D2 \wedge D3 \wedge \ldots\ldots \wedge Dk) \rightarrow D_{k+1}$

3 Forms of Horn Clauses

1) At most 1 positive literal

$\neg D1 \lor \neg D2 \lor \neg D3 \lor \ldots\ldots \lor \neg Dk \lor Dk+1$

Or

$(D1 \land D2 \land D3 \land \ldots\ldots \land Dk) \rightarrow Dk+1$

Or

$Dk+1 \leftarrow (D1 \land D2 \land D3 \land \ldots\ldots \land Dk)$ (RHS implies LHS)

2) Only 1 Positive literal (Nothing implies it)

$\rightarrow Dk+1$

Or

$Dk+1 \leftarrow$

3) 0 positive literals (All negative)

$(D1 \land D2 \land D3 \land \ldots\ldots \land Dk) \rightarrow$

Or

$\leftarrow (D1 \land D2 \land D3 \land \ldots\ldots \land Dk)$ (Query / Goal)

Horn Clause and Prolog

- Form 1 is RULE in Prolog

a:- b,c,d Clause form is $\neg b \wedge \neg c \wedge \neg d \vee a$

- Form 2 is FACT in Prolog

c:- Clause form is c

d:- Clause form is d

- Form 3 is QUERY in Prolog

:- a Clause form is $\neg a$

Sample Statements and Horn Clause form

Bob is Fred’s father **father(Bob, Fred)**
Sue is Fred’s mother **mother(Sue, Fred)**
Barbara is Fred’s sister **sister(Barbara, Fred)**
Jerry is Bob’s father **father(Jerry, Bob)**

And the following **rules**:

A person’s father’s father is the person’s grandfather
A person’s father or mother is that person’s parent
A person’s sister or bother is that person’s sibling
If a person has a parent and a sibling, then the sibling has the same parent

These might be captured in **first-order predicate calculus** as:

x, y, z : if father(x, y) and father(y, z) then grandfather(x, z)
x, y : if father(x, y) or mother(x, y) then parent(x, y)
x, y : if sister(x, y) or brother(x, y) then sibling(x, y) and sibling(y, x)
x, y, z : if parent(x, y) and sibling(y, z) then parent(x, z)

We would rewrite these as

grandfather(x, z) ⊂ father(x, y) and father(y, z)
parent(x, y) ⊂ father(x, y)
parent(x, y) ⊂ mother(x, y)