Knowledge Representation and Inferencing in AI

Susmita Ghosh
Computer Science & Engg. Department
Jadavpur University, Kolkata, India
email: susmitaghoshju@gmail.com

AI Production System

Most AI systems show a separation between standard computational components of Data, Operation and Control

Major elements: a global database (3x3 matrix…), a set of production rules (if-then..), a control system

Control system: selects rules and keeps track of those sequences of rules already tried and the databases they produced

AI Production System

Control strategy:

Irrevocable: No reconsideration of a rule

Tentative: two types: Backtracking (a backtracking point is selected) and graph search control (provision is made for keeping track of the effects of several sequences of rules simultaneously)

Procedure Production

```
1. DATA <- Initial database
2. until DATA satisfies the termination condition
   do:
3. Begin
4. Select some rule, R, from the set of Rules that
   can be applied to DATA
5. DATA <- Result of applying R to DATA
6. end
```

FOPL

A formal language in which a wide variety of statements can be expressed

- Define the language
- How it is used to represent statements
- How inferences are drawn
- How statements are deduced

Introduction to Predicate Calculus

Syntax: alphabets of symbols and how they are put together to form legitimate expression (*wff*)

Elementary components:
predicate symbol, variable symbol, function symbol, constant symbol

Alongwith parenthesis, brackets, commas
connectives
Quantifiers

WRITE(Nilsson, Principles……)
For all/ There exists WRITE(x,y)
John's brother and John's sister are sibling to each other
SIBLING (brother(JOHN), sister(JOHN))

The House is Yellow.
John lives in a yellow house.
Connectives: AND, OR, Not
LIVES(JOHN, H1) ^ COLOR(H1, YELLOW)
========================
And: ^ Not: ~ /¬ Or: v
(∀x) ELEPHANT(x) => COLOR (x, GREY)
(∃y) PERSON(y) ^ **WROTE(y, GITA)**

Predicate Calculus

John Lives in a Yellow House

LIVE(JOHN, HOUSE1) ^ COLOR (HOUSE1, YELLOW)
HOUSE(JOHN,YELLOW)

John Plays Chess or Badminton
PLAYS(JOHN, CHESS) U PLAYS(JOHN,BADMINTON)

If the car belongs to John, then it is yellow
BELONGS (JOHN, CAR1) => COLOR (CAR1, YELLOW)

A => B   is equivalent to ~A U B

Proposition Calculus: as *terms* we never used variables

~ (~X)

~(X1 ^ X2)   ~X1 U ~X2

X1 ^ (X2 U X3)

X1 ^ X2

Contrapositive: X1 => X2   ~X2 => ~X1

~(∃y) P(y)

(∀x) [P(x) and Q(x)]

(∀x) [P(x)] and (∀y) [Q(y)]

Predicate Calculus

All elephants are Grey.

$(\forall x) ELEPHANT(x) => COLOR(x, GREY)$

There is a person who wrote Gita.

$(\exists x)\ PERSON(x) \wedge WROTE\ (x, GITA).$

$\sim(\forall x)\ P(x) \quad = (\exists x) \sim P(x)$

$(\forall x)\ [P(x \wedge Q(x)] \quad (\forall x)\ P(x) \wedge (\forall y)\ Q(y)$

$(\exists x)$ ….

$(\forall x)\ P(x)$ and $(\forall y)\ P(y)$

x, y – bound var--- > dummy var

This is FOPC

Equivalent wffs

For every set x, there is a set y, such that cardinality of y is greater than that of x.

$(\forall x) \{Set(x) \Rightarrow (\exists y) (\exists u) (\exists v)$
$[Set(y) \wedge Card(y, u) \wedge Card(x, v) \wedge Greater(u,v)]\}$

Our Aim

Given the **Database**

1. Wff1
2. Wff2
3. Wff3
4. Wff4

5. Prove wff5

a)Wff1, wff2 ---- wffx (which two can be combined, is supported by **Operators/ Rules/--- here , Rules of inference**)
b)Wff3, wff4 --- wffy
…………..
m)Wff7, wffx ---- wff5 /
n) Wffx, wffy --- wff5 say, *wff5 is my derived wff*
(whether I will first apply a) then b) or the opposite or some other… will be governed by **Control Strategy**)

Rules of Inference, Theorems, Proofs

Rules of Inference can be applied to certain wffs and sets of wffs to produce new wffs.

1.Modus ponens

Produce W2 from (i) W1 and (ii) W1=>W2

2.Universal specialization

Produce W(A) from (i) (∀x) W(x)

3. Using both,

Produces W2(A) from

(i) (∀x) [W1(x) => W2(x)] and (ii) W1(A)

Unification

Unification is done to match certain sub expressions in a given set of expressions.

Let our database contains:
(1)(∀x) [W1(x) => W2(x)]
(2) W1(A)
From 1 and 2 we can derive W2(A).
To do that, we substitute A for x which makes W1(A) and W1(x) identical

Unification: Finding substitution of terms for variables to make expressions identical
(e.g., E1 = W1(A), E2 = W1(x)
To make, E1 = E2, I need some substitution, here it is {A/x})

I can substitute a **variable** by a **term** , where,
terms: var/ constant/ function

Constraints: x by A --- throughout ~~Replace x by f(x)~~

Unification: substitution instance, composition

Substitution instance: of an expression is obtained by substituting terms for variables in that expression.   E =   P[x,f(y),B]

E = P[x,f(y),B]

E1 = P[z,f(w),B]   s1 = { }
E2 = P[x,f(A),B]   s2 = { }
E3 = P[g(z),f(A),B]   s3 = { }
E4 = P[C,f(A),B]   s4= {}

s = {t1/v1, t2/v2,…, ti/vi,…, tn/vn}

To denote a substitution: Es
P[z,f(w),B] = P[x,f(y),B] s1
Composition of two substitutions : Es1s2
s1   s2   ??
{g(x,y)/z} {A/x, B/y, C/w, D/z}
E = (x, y, z) ------ s = {g(A,B)/z, A/x, B/y, C/w}
Substitution obtained by applying s2 to the terms of s1 and then adding any pairs of s2 having variables not occurring among variables of s1.

Unification: substitution instance, composition

Substitution instance: of an expression is obtained by substituting terms for variables in that expression.   E = P[x,f(y),B]

E = P[x,f(y),B]

E1 = P[z,f(w),B]   s1 = {z/x, w/y}   alphabetic variant (E1 = Es1)
E2 = P[x,f(A),B]   s2 = {A/y}
E3 = P[g(z),f(A),B]   s3 = {g(z)/x, A/y}
E4 = P[C,f(A),B]   s4= {C/x, A/y}   ground instance (E4= Es4)

s = {t1/v1, t2/v2,…, ti/vi,…, tn/vn}

To denote a substitution: Es
P[z,f(w),B] = P[x,f(y),B] s1
Composition of two substitutions : Es1s2
s1   s2 ??
{g(x,y)/z} {A/x, B/y, C/w, D/z}
E = (x, y, z) ------ s = {g(A,B)/z, A/x, B/y, C/w}
Substitution obtained by applying s2 to the terms of s1 and then adding any pairs of s2 having variables not occurring among variables of s1.

Unification: substitution instance, composition

Es' = Es1s2 = Es2s1 ??? Is it Correct ?
s1s2 = s2s1 wrong (commutative property will not hold here)
(s1,s2)s3 = s1(s2s3) correct (Associative)

Unifiable:
{E1, E2, E3….}
A set {Ei} s
E1s = E2s = E3s …… Ens = E

{Ei}is unifiable, s is unifier

s = {A/x, B/y} unifies
{Ei} = {P[x,f(y),B], P[x,f(B),B]}= P[A,f(B),B]
---- is s the simplest unifier?

s is NOT mgu
g of {Ei}
s {Ei} {Ei}s
{Ei}s = {Ei}gs'

Sets of Literals | Most Common Substitution instances

i. {P(x), P(A)} — P(A) {A/x}

ii. {P[f(x),y,g(y)], P[f(x),z,g(x)]} — P[f(x), x, g(x)] {x/y, x/z}

iii. {P[f(x,g(A,y)), g(A,y)], P[f(x,z),z]} — P[f(x,g(A,y)), g(A,y)]

The disagreement set of E is the set D obtained by comparing each symbol of all expressions in E from left to right and extracting from E the subexpression whose first symbols do not agree.

E = {P(f(x), g(y), a), P(f(x), z, a), P (f(x), b, h(u))} --- D={}

Unification Algorithm

1. Set k = 0 , mgu_k = {}
2. If the set $Emgu_k$ is a singleton then stop; mgu_k is an mgu of E. Otherwise, find the disagreement set D_k of $Emgu_k$
3. If there is a var v and term t in D_k such that v does not occur in t, put mgu_{k+1} = mgu_k{t/v}, set k = k+1, and return to step 2. Otherwise, stop, E is not unifiable.

E = {P(x,z,y), P(w,u,w), P(A,u,u)}

mgu ??

Resolution

An important *rule of inference* that can be applied to certain class of wffs, called clauses

A clause is defined as a wff consisting of a disjunction of literals.

The resolution process, when it is applicable, is applied to a pair of parent clauses to produce a derived clause.

Any Predicate calculus wff can be converted to a set of clauses.

Clause form

- A clause is a disjunction ("**or**") of zero or more literals, some or all of which may be negated
- Example:
  sinks(X) ∨ dissolves(X, water) ∨ ¬denser(X, water)
- Notice that clauses use only "or" and "not"—they do not use "and," "implies," or either of the quantifiers "for all" or "there exists"
- The impressive part is that *any* predicate calculus expression can be put into clause form
  - Existential quantifiers, ∃, are the trickiest ones

The magic of resolution

- Here's how resolution works:
  - transform each of your facts into a particular form, called a clause
  - apply a *single rule*, the resolution principle, to a pair of clauses
    - Clauses are closed with respect to resolution--that is, when you resolve two clauses, you get a new clause
  - add the new clause to your fact base
- So the number of facts grows linearly
  - You still have to choose a pair of facts to resolve
  - You never have to choose a rule, because there's only one

The resolution principle

- Here it is:
  - From X ∨ someLiterals
    and ¬X ∨ someOtherLiterals
    ----
    conclude: someLiterals ∨ someOtherLiterals
- That's all there is to it!
- Example:

  broke(Bob) ∨ well-fed(Bob)
  ¬broke(Bob) ∨ ¬hungry(Bob)
  ----
  well-fed(Bob) ∨ ¬hungry(Bob)

A common error

- You can only do *one* resolution at a time
- Example:

  broke(Bob) ∨ well-fed(Bob) ∨ happy(Bob)
  ¬broke(Bob) ∨ ¬hungry(Bob) ∨ ¬happy(Bob)
- You can resolve on broke to get:
  - well-fed(Bob) ∨ happy(Bob) ∨ ¬hungry(Bob) ∨ ¬happy(Bob) ≡ T
- Or you can resolve on happy to get:
  - broke(Bob) ∨ well-fed(Bob) ∨ ¬broke(Bob) ∨ ¬hungry(Bob) ≡ T
- Note that both legal resolutions yield a tautology (a trivially true statement, containing (X ∨ ¬X), which is correct but useless
- But you *cannot* resolve on both at once to get:
  - well-fed(Bob) ∨ ¬hungry(Bob)

Contradiction

- A special case occurs when the result of a resolution (the resolvent) is empty, or “NIL”
- Example:

```
  hungry(Bob)
¬hungry(Bob)
----------------
NIL
```

- In this case, the fact base is inconsistent
- This will turn out to be a very useful observation in doing resolution theorem proving

Converting to CNF

Converting sentences to CNF

$(\forall x)\{P(x) \rightarrow \{(\forall y)[P(y) \rightarrow P(F(x,y))] \wedge \neg(\forall y)[Q(x,y) \rightarrow P(y)]\}\}$

1. Eliminate all $\leftrightarrow$ connectives

   $(P \leftrightarrow Q) \Rightarrow ((P \rightarrow Q) \wedge (Q \rightarrow P))$

2. Eliminate all $\rightarrow$ connectives

   $(P \rightarrow Q) \Rightarrow (\neg P \vee Q)$

3. Reduce the scope of each negation symbol to a single predicate

   $\neg\neg P \Rightarrow P$

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

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

   $\neg(\forall x)P \Rightarrow (\exists x)\neg P$

   $\neg(\exists x)P \Rightarrow (\forall x)\neg P$

4. Standardize variables: rename all variables so that each quantifier has its own unique variable name

Converting sentences to clausal form: Skolem constants and functions

5. Eliminate existential quantification by introducing Skolem constants/functions

   $(\exists x)P(x) \Rightarrow P(C)$

   C is a Skolem constant (a brand-new constant symbol that is not used in any other sentence)

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

   since $\exists$ is within the scope of a universally quantified variable, use a **Skolem function F** to construct a new value that **depends on** the universally quantified variable

   F must be a brand-new function name not occurring in any other sentence in the KB.

Converting sentences to clausal form

6. Remove universal quantifiers by (1) moving them all to the left end; (2) making the scope of each the entire sentence; and (3) dropping the “prefix” part

(Prenex form :Prefix Matrix) Ex: $(\forall x)P(x) \Rightarrow P(x)$

7. Put matrix into conjunctive normal form (conjunction of disjunctions) using distributive and associative laws repeatedly

x1 or (x2 and x3) by (x1 or x2) and (x1 or x3)

$(P \wedge Q) \vee R \Rightarrow (P \vee R) \wedge (Q \vee R)$

$(P \vee Q) \vee R \Rightarrow (P \vee Q \vee R)$

8. Split conjuncts into separate clauses

(x1 and x2) {x1, x2}

9. Standardize variables so that each clause contains only variable names that do not occur in any other clause

Produce Resolvent from 2 Parent Clauses

Parent Clauses	Resolvent	Comment
P and ~P or Q	Q	Modus ponens
(P or Q) and (~P or Q)	Q	Merge operation
(P or Q) and (~P or ~Q)	Q or ~Q and P or ~P	tautologies
P and ~P	NIL	empty clause; sign of contradiction
(~P or Q) and (~Q or R)	~P or R	Chaining

Resolution Refutation System

- One type of theorem proving system.
- Designed to produce proofs by contradictions/ refutations

We have a set, S, of wffs from which we wish to prove some goal wff, W

1. Negate W and add it to S --- S U{~W}
2. Convert new S to a set of clauses

--- attempt to derive a contradiction represented by NIL

Key idea: If W logically follows from S, the set S U{~W} is unsatisfiable.

Therefore, if empty clause NIL is produced from S U{~W}, then W logically follows from S.

Production Systems for Resolution Refutations

- Let S be the set of clauses (base set)

Algo:

- 1. Clauses = S
- 2. until NIL is a member of Clauses do:
- 3. begin
- 4. Select two distinct resolvable clauses, Ci and Cj, from Clauses
- 5. Compute a resolvent r_{ij} to Clauses
- 6. Clauses = The set produced by adding r_{ij} to Clauses
- 7. end

Control Strategies for Resolution Refutation

- The decision about which two clauses in Clauses to resolve and which resolution of these clauses to perform, are done irrevocably by the control strategy.
- Control strategy uses derivation graph

  nodes – clauses

Initially, for every clause in the base set, a node exists

From Ci and Cj create rij →

$$\begin{tikzcd}
Ci & & Cj & & Ck \\
& rij \arrow[ul, dash] \arrow[ur, dash] & & & \\
& & & rijk \arrow[ull, dash] \arrow[uur, dash] &
\end{tikzcd}$$

In refutation tree, root is NIL

A control strategy for a refutation tree is said to be complete if its use results in a procedure that will find a contradiction (eventually), whenever one exists.

Example

1. Whoever can read is literate (for all x) [R(x) -> L(x)]
2. Dolphins are not literate (for all y) [D(y) -> ~ L(y)]
3. Some dolphins are intelligent (there exists z) [D(z) and I(z)]

Prove that,

- Some who are intelligent cannot read (there exists w) [I(w) and ~R(w)]

- $$\begin{tikzcd}
\sim R(x) \text{ or } L(x) \arrow[dr, dash] & & \sim D(y) \text{ or } \sim L(y) \arrow[dl, dash, "\{x/y\}"] & \\
& \sim R(x) \text{ or } \sim D(x) \arrow[dr, dash, "\{A/x\}"'] & & D(A) \arrow[dl, dash] \\
& & \sim R(A) &
\end{tikzcd}$$

Sound and Complete

- Resolution is a sound rule of inference---- the resolvent of a pair of clauses also logically follows from the pair of clauses

- When resolution is used in a special kind of theorem proving system, and called a refutation system, it is also complete ----- Every wff that logically follows from a set of wffs, can be derived from that set of wffs using resolution refutation

Control Strategies for Resolution Refutation

- Breadth First Strategy

Compute all first level resolvents (between two clauses in the base set), then second level… and so on

ith level resolvent → deepest parent is an (i-1)th level resolvent

Complete but inefficient

- Linear Input Form Strategy

Each resolvent has at least one parent belonging to the base set. At subsequent levels, it reduces the number of clauses produced

Not complete

Example:

~Q(x) U ~P(x) , Q(y) U ~P(y), ~Q(w) U P(w), Q(u) U P(A)

Control Strategies for Resolution Refutation

- Set of Support Strategy

At least one parent of each resolvent is selected from negation of goal wff or from their descendants.

Complete

- Unit Preference Strategy

Modification of set of support; instead of filling out each level in breadth first, try to select a single literal clause (unit) to be a parent in resolution

Complete, typically increases efficiency

- Ancestry Filtered Form Strategy

Each resolvent has a parent that is either in the base set or ancestor of other parent

- Combination of strategies

Set of support with either linear input form/ ancestry filtered form is common

Extracting Answers from Resolution Refutation

- If Fido goes wherever John goes, and if John is at School, where is Fido?

1. Append to each clause arising from the negation of goal wff, its own negation
2. Following the structure of the refutation tree, perform the same resolutions as before until some clause is obtained at the root
3. Use the clause at the root as an answer statement

Thank You