First Order Predicate Logic

Topics of this lecture

- Predicate logic
- Term and logic formula
- Clausal form/Conjunctive canonical form
- Standardization of logic formula
- Unification and resolution
- Horn clause and Prolog

- 述語論理
- 項と論理式の定義
- 節形式/連言標準形
- 論理式の標準化
- 単一化と導出
- ホーン節とプロログ

Predicate logic (述語論理)

- In propositional logic, the smallest unit is a **proposition**.
- Using propositional logic, it is difficult to represent knowledge like “For anyone, if he/she is a Japanese, he/she is a human”.
- In predicate logic, the **subject** and **predicate** are further divided. For example,
  - Daisuke is a Japanese → J(Daisuke)
  - Daisuke is a human → H(Daisuke)
- This division is convenient to represent knowledge that is generally or sometime useful.

1st order predicate logic (第一階述語論理)

- Example of ***generally*** useful (valid) knowledge
  - $\forall x \; J(x) \Rightarrow H(x)$:

    For any x, if x is a Japanese, x is a human.
- Example of knowledge that is ***sometimes*** useful
  - $\exists x \; H(x) \Rightarrow J(x)$

    For some x, x is a human, x is a Japanese.
- In the **1st order predicate logic**, variables can only be "individuals" in the ***universe of discourse*** (対象領域).
- For high order predicate logic, predicates and functions can also be variables.

Terminologies

- Domain or universe of discourse
  - Denoted by D. Examples include
  - Sets of students, human, living being.
  - Sets of prime numbers, integers, real numbers.
  - Sets of machines, robots, systems.
- Individual constant (個体定数) in D:
  - a, b, c, Taro, Hanako
  - Used in the same way as in propositional logic
- Individual variable(個体変数) in D:
  - x, y, z
  - Used to represent something not yet fixed.

Terminologies

- ***Functional*** symbol (関数記号):
  - Map from D to D: find individual(s) from given individual(s).
  - Ex1: y=**mother**(x): Find the mother of x, and put the result into y.
  - Ex2: **brother**(x): Find the brother(s) of x.
- ***Predicate*** symbol (述語記号):
  - Map from D to {True, False}: from individuals to a logic value
  - Ex1: Human(x): True if x is a human.
  - Ex2: Mother(x, y): True if y is the mother of x.
- Example 3.5 (p. 49)
  - a="Chieko", b="Makoto";
  - Known: *Chieko is the mother of Makoto*
  - Mother(b,a) is True;
  - mother(b) is a="Chieko".

Terminologies

- Logical symbol (論理記号):
  - ∧、∨、¬、⇒,⇔
  - The same as used in propositional logic.
- ***Quantifier*** (限量記号):
  - Here, we consider only the **universal quantifier** (全称記号)"∀" and the **existential quantifier** (存在記号)"∃".
  - Ex1: ∀x H(x)⇒C(x):
    - For any x, if x is a human, x is "cybernetic".
  - Ex2: ∃x H(x)⇒G(x):
    - There exists an x, if x is a human, x is "genius".

Definition of ***term***

(1) Any individual (constant or variable) in D is a term (項).

(2) If t1,t2,…,tn are terms and f is an n argument function, f(t1,t2,…,tn) is also a term.

(3) Only expressions obtained using (1) and (2) are terms.

- Examples of term:
  - Individual constants: Taro, Jiro, Chieko
  - Individual variables: x, y, z in D
  - Function: mother(Makoto), brother(Jiro)

Definition of atomic formula (素式)

- If t1,t2,…,tn are terms, and P is a predicate symbol representing the relation between these terms, P(t1,t2,…,tn) is an atomic logic formula.
- Examples of atomic formulas
  - Human(a): True if a is human
  - Parent（b, mother(a)）): True if mother(a) is (also) the parent of b.
  - Mother(a, mother(a)) is always True.
- Atomic formulas are the smallest units in the predicate logic.

Definition of logical formulas

(1) An atomic formula is a logic formula.

(2) If P and Q are logic formulas, $\neg P$, $P \wedge Q$, $P \vee Q$, $P \Rightarrow Q$, and $P \Leftrightarrow Q$ are logic formulas.

(3) If P is a logic formula, and x is an individual variable, $\forall x\, P$ and $\exists x\, P$ are logic formulas.

(4) Only expressions defined by (1), (2), and (3) are logic formulas.

As in propositional logic, logic formulas defined above are called well-formed formulas (wff).

The clausal form

- An atomic formula and its negation are called **literals**. The disjunction (OR) of the literals is called a **clause**.
- If Ci (i=1,2,…,n) are clauses, the closed formula (閉式) given below is called a **clausal form**:

$$Qx_1Qx_2\ldots Qx_m[C_1 \wedge C_2 \wedge \ldots \wedge C_n]$$

- Here, Q is a quantifier symbol, and $x_1, x_2, \ldots, x_m$ are individual variables contained in the logic formula.
- A "**closed formula**" is a formula in which all variables are bounded (束縛される) by the quantifiers. That is, there is no free variables (自由変数).
- The logic formula in the bracket is called a matrix (母式).

Process for converting a logic formula to the clausal form

- Any predicate logic formula can be converted to the clausal form as follows:
  - Remove the equivalence and implication symbols;
  - Move the negation symbol immediately before the atomic formula;
  - Remove ambiguities of the variables;
  - Remove the existential quantifier $\exists$ using Skolem constant or Skolem function;
  - Remove all universal quantifiers;
  - Use distributive laws.

Process for converting a logic formula to the clausal form

- Remove the equivalence and implication symbols
  - $P \Rightarrow Q = \neg P \lor Q$

- Move the negation symbol immediately before the atomic formula
  - $\neg(\neg P) = P$
  - $\neg(P \land Q) = \neg P \lor \neg Q$, $\neg(P \lor Q) = \neg P \land \neg Q$
  - $\neg \forall x P(x) = \exists x(\neg P(x))$, $\neg \exists x P(x) = \forall x(\neg(P(x))$

- Remove ambiguities of the variables
  - $\forall x P(x) \lor \forall x Q(x) = \forall x P(x) \lor \forall y Q(y)$

Process for converting a logic formula to the clausal form

- Remove the existential quantifier ∃ using ***Skolem constant*** or ***Skolem function***
  - $\exists x\, P(x) = P(s_1)$
  - $\forall x \exists y\ P(x,y) = \forall x\, P(x,s_2(x))$

  - Ex1 : $\exists x\, Fly(x) = Fly(X)$
    - There is an x, x can fly ⬄ For some X, X can fly.

  - Ex2 : $\forall x \exists y\, Love(x,y) = \forall x\, Love(x,love(x))$
    - For any x, there is a y, y loves x ⬄
      For any x, Y=love(x) loves x
    - Pay attention to the difference between Love() and love().

Set of clauses

- A logic formula in clausal form is equivalent to a set of clauses. In turn, this set can be considered a ***knowledge base*** in which each clause is a “rule”.

1	$\neg H(x_1) \vee M_1(x_1)$
2	$\neg B_1(x_2) \vee M_1(x_2)$
3	$\neg M_1(x_3) \vee \neg E(x_3) \vee C(x_3)$
4	$\neg M_1(x_4) \vee \neg S_1(x_4) \vee \neg S_2(x_4) \vee C(x_4)$
5	$\neg C(x_5) \vee \neg B_2(x_5) \vee \neg B_3(x_5) \vee L(x_5)$
6	$\neg C(x_6) \vee \neg B_2(x_6) \vee \neg M_2(x_6) \vee F(x_6)$
7	$H(a)$
8	$B_2(a)$
9	$B_3(a)$

A generalized rule for reasoning

- Suppose there are two clauses C1 and C2 which contain a literal P and its negation ¬P, respectively.
- From C1 and C2, we can derive a new clause R which contains all literals of C1 and C2 except P and ¬P.
- C1 and C2 are called the **parent clauses**, and R is called the **resolvent clause**.
- The above process is called **resolution** **(導出**）.

Example 3.8 p. 55

- C1=¬S(a)∨H(a);
- C2=¬H(x)∨M(x)
- R=¬S(a)∨M(a)

S(x): x is a student;
H(x): x is a human;
M(x): x is mortal

The left resolution process is actually equivalent to syllogism（三段論法）.

Here, we have substituted a into x. Through this substitution, we can convert H(a) and H(x) into the same form. This is called **unification** (単一化）.

More about unification

- The purpose of unification is to enable “pattern matching” between two compliment literals.
- If the two compliment literals are the same from the beginning, it is not necessary to conduct unification.
- In the process of automatic inference (reasoning), one of the literals is often more “general” than its negation.
- Unification often makes the literals more specific.

Examples 3.9 and 3.10 pp. 55-56

- Unification of F(x,q) and F(p,y):
  - F(x,q) and F(p,y) can be unified by defining x=p and y=q.
  - This unification can be denoted by {x/p,y/q}, and it is called a unifier.

- Unification of Q(f(b),x) and Q(y, a)
  - Q(f(b),x) and Q(y, a) can be unified by defining x=a and y=f(b)
  - The unifier is {x/a,y/f(b)}.

Prove by refutation (反駁証明）

- To prove whether a given formula is a theorem or not, we will have to show that the formula is true for all possible individuals in D. This is usually very difficult.
- A more efficient way is to **prove by refutation**.

That is, if we add the negation of the formula into the set of clauses, and if we can derive a contradiction, the formula cannot be refused and it must be true.

Example 3.11 pp. 56-57

- Suppose that a set of clauses is given in Table 3.8. Prove $E(a) \Rightarrow L(a)$.

- Proof: The formula to prove is actually $P = \neg E(a) \vee L(a)$, and its negation is $\neg P = E(a) \wedge \neg L(a)$.
- Since $\neg P$ is the AND of two formulas, we should add both $E(a)$ and $\neg L(a)$ to the set of clauses, in order to conduct proof through refutation.

Example 3.11 pp. 56-57

- C1：¬L(a) (*added clause*)
- C2：¬C(x5)∨¬B2(x5)∨¬B3(x5)∨L(x5) (the 5th clause)
- R1: ¬C(a)∨¬B2(a)∨¬B3(a) unifier: {x5/a}

- C1: R1 obtained above
- C2: B3(a) （the 9th clause）
- R2：¬C(a)∨¬B2(a) (unification not needed)

- C1: R2 obtained above
- C2: B2(a) （the 8th clause）
- R3: ¬C(a) (unification not need)

- C1：R3 obtained above
- C2：¬M(x3)∨¬E(x3)∨C(x3) （The 3rd clause）
- R4：¬M(a)∨¬E(a) unifier：{x3/a}

Example 3.11 pp. 56-57

- C1：R4 obtained above
- C2：E(a) （*added clause or fact*)
- R5：¬M(a) (unification not needed)

- C1：R5 obtained above
- C2：¬H(x1)∨M(x1) （the 1st clause）
- R6：¬H(a) unifier: {x1/a}

- C1：R6 obtained above
- C2：H(a) （The 7th clause）
- R7: Φ *(unification not needed)*

In fact, this proof is similar to backward reasoning. That is, to verify that an animal is lion if it eats meat, based on the set of clauses, which include both the knowledge and the observations.

Φ is the empty clause, means a contradiction!

Horn Clause (ホーン節)

- A clause can be generally represented as follows:

$$\neg A1 \lor \neg A2 \lor \ldots \lor \neg Am \lor B1 \lor B2 \lor \ldots \lor Bn$$

- If n <= 1, the clause is called a **Horn clause**; if n=1, the clause is a **definite clause**; if n=0, the clause is a **goal clause**; if m=0, the clause is a fact.
- Using implication, a Horn clause equals to

$$A1 \land A2 \land \ldots \land Am \Rightarrow B$$

- To simplify the proof process, it is convenient to ***put the consequence to the left side*** (to improve the efficiency of formal proof).

Horn Clause

- If the premise is the disjunction of several literals, the formula can be separated to form several Horn clauses. For example:

  $A1 \vee A2 \Rightarrow B \quad \rightarrow \quad A1 \Rightarrow B$ and $A2 \Rightarrow B$

- On the other hand, if there are several consequences for a premise, that is, the consequence is the conjunction of several literals, the formula can be separated into several Horn clauses. For example:

  $A \Rightarrow B1 \wedge B2 \quad \rightarrow \quad A \Rightarrow B1$ and $A \Rightarrow B2$

Selective linear resolution for definite clause (SLD:選択的線形導出)

1) Assign the goal clause (to be proved) to the first parent clause C1.
2) Find a C2 from the set of clauses that contains a positive literal unifiable with the left literal of C1.
3) Obtain a resolvent clause R from C1 and C2. If R is the empty clause, stop; otherwise, put R to C1 and return to 2).

Capability of the linear approach

- In general proof based on the above linear strategy is not complete.
- That is, even if a goal clause is not satisfiable, we may not get the empty clause when the program stops.
- To solve the problem, it is necessary to use more general search algorithms like depth-first or breadth-first search.

The process of formal proof is actually search. SLD is nothing but a heuristic for efficient search.

Programing based on Prolog

- Prolog is a non-procedural programming language. It belongs to logic programing language.
- In fact, Prolog is the short for ***programming in logic***.
- Prolog was created around 1972 by A. Colmerauer with Philippe Roussel.
- In Prolog, a Horn clause is represented by
  - B:-A1,A2,…,An
  - This is equivalent to $A1 \wedge A2 \wedge \ldots \wedge An \Rightarrow B$
- The reason to put B on the left is to improve the reasoning efficiency.
- "$\Rightarrow$" is replaced by ":-", and "$\wedge$" is replaced by "," to make it easier to input the symbols using keyboard.

An example of Prolog program

Table 3.10, p. 60

Program	Meaning
parent(sofu,otosan).	sofu is the parent of otosan.
parent(msofu,okasan).	msofu is the parent of okasan.
parent(otosan,miho).	otosan is the parent of miho.
parent(otosan,taro).	otosan is the parent of taro.
parent(otosan,jiro).	otosan is the parent of jiro.
parent(X,Y):-married(Z,X),parent(Z,Y).	If Z and X are husband and wife, AND Z is the parent of Y, X is the parent of Y.
married(otosan,okasan).	otosan and okasan are husband and wife.
married(msofu,msobo).	msofu and msobo are husband and wife.
married(sofu,sobo).	sofu and sobo are husband and wife.
ancestor(X,Y):-parent(X,Y).	If X is the parent of Y, X is the ancestor of Y.
ancestor(X,Y):-parent(X,Z),ancestor(Z,Y).	If X is the parent of Z, AND Z is the ancestor of Y, X is the ancestor of Z.

To run the test program

```
swipl
% omitted several lines here

1 ?- [test].
true.

2 ?- parent(X,taro).
X = otosan ;
X = okasan ;
false.

3 ?- ancestor(sofu,X).
X = otosan ;
X = miho ;
X = taro ;
X = jiro ;
false.
```

Save the prolog program given in the previous page to ‘test.pl’, and then run ‘swipl’ in the university environment.
If you use your own PC, please download the free software first.

URL: http://www.swi-prolog.org/pldoc/man?section=quickstart

Homework for lecture 8 (1)
(submit to the TA during the exercise class)

- Solve Problem 3.5 given in p. 59 in the textbook.
- Table 3.8 is a set of Horn clauses.
- Prove L(a) using the SLD strategy.

Homework for lecture 8 (2)

- Write a Prolog programming based on the following set of clauses:

$p(X,Y) \lor \neg q(X) \lor \neg r(Y)$
$q(X) \lor \neg s(X)$
$r(Y) \lor \neg t(Y)$
$s(a)$
$t(b)$

Note 1: $A \Rightarrow B = (\neg A \lor B)$
Note 2:
$g(X) \land h(Y) \Rightarrow f(X, Y)$ becomes
f(X, Y):- g(X), h(Y)   in Prolog.

- Save your program into “prog_08_1.pl”, and prove p(a, b) using this program.
- Try the program given in p. 29 by saving the program into “prog_08_02.pl”.

Quizzes of today

- In 1st order predicate logic, [________] can be a variable.
- In predicate logic, the return value of a function is [______________________].
- To specify the scope of a variable, we use quantifiers. The universal quantifier is denoted by [________].
- In predicate logic, we have “terms” and “atomic formulas”. The smallest unit of a logic formula is [_________________].

- We can use [___________________] constant to remove existential quantifiers in a formula.
- Unification is used to make two literals the same form. After unification, we can derive a resolvent clause from two [__________________] clauses.
- A Horn clause is a clause with [______________________________] positive literal.
- Prolog is a short of [______________________________].